HEURISTIC SCHEDULING PROCEDURES TO ACHIEVE WORKLOAD BALANCE ON PARALLEL PROCESSORS

A THESIS

Presented to

The Faculty of the Division of Graduate

Studies and Research

by

Emett Robert White

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Industrial Engineering

Georgia Institute of Technology

November 1973

HEURISTIC SCHEDULING PROCEDURES TO ACHIEVE WORKLOAD BALANCE ON PARALLEL PROCESSORS

Approved:

Richard H. Deane, Chairman

D. C. Montgomery

R. G. Parker

Date approved by Chairman: 1 Nov 1973

ACKNOWLEDGMENTS

The author wishes to express his sincere gratitude to his thesis advisor, Dr. Richard H. Deane for his invaluable guidance, encouragement and constructive criticism which made possible the successful completion of this research.

Special thanks must also be given to Drs. R. G. Parker and Douglas C. Montgomery for their comments and encouragement as members of the thesis committee.

Appreciation is also due to Peggy Weldon for her skillful assistance in the preparation of this thesis.

A very special expression of thanks is due the author's wife, Ann, for her love, encouragement, and companionship throughout the conduct of this research.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF ILLUSTRATIONS

SUMMARY

This research examines several heuristic scheduling procedures designed to generate minimal cost balanced workloads on parallel processors. The problem involves the scheduling of M independent batch-type jobs with sequence dependent set-up costs, on N parallel processors. It is assumed that sequence dependent change-over costs are not identical for each processor and that all jobs are available at some arbitrary time zero. The objective is to maintain each processor workload assignment T_n within a specified limit (D) of the average assigned machine workload.

This problem is characterized by a large number of solutions which satisfy the basic balancing condition thus allowing it to be defined as a constrained optimization problem. The objective becomes the minimization of total change-over cost subject to the balance constraint. A branch and bound algorithm for this problem is presented and programmed for a digital computer so that evaluation of the heuristic procedures can be made on the basis of change-over cost. Further comparative evaluation of the heuristic procedures is based on several defined balance measures.

CHAPTER I

INTRODUCTION

In most industrial scheduling situations where mathematical or graphical procedures could be used, optimal solution procedures are hampered by environmental and/or economic constraints placed upon the amount of effort expended on a particular problem in any given instance. In some instances it may be the availability of electronic computers of sufficient capacity; in others it may be the incremental costs associated with the requirement for additional trained personnel to implement the scheduling system. As the volume of research in the area of scheduling parallel processors grows, it is apparent that optimal solution procedures suffer the same computational difficulties associated with the single processor problem and which, for the reasons just mentioned, prevent their use in actual job-shop situations. Therefore, it becomes important to first investigate alternate solution procedures, such as heuristics, which can provide either near optimal or, at least, significantly better than randomly generated, schedules to meet the goals of management.

This research is conducted with the intent of examining several heuristic procedures in the scheduling of batch type jobs with sequence-dependent change-over costs on parallel processors when a balancing restriction is imposed. Additionally, a branch and bound algorithm will be formulated to obtain, for comparative purposes, an optimal solution to this problem.

The criteria of workload balancing is receiving an increasing amount of attention. Its importance arises from the observation that schedules derived from procedures based upon various alternative criteria, such as the satisfaction of due dates, may result in unbalanced schedules. If we recognize that average through-put time represents perfect balance, then workload completion times which precede or exceed this time result in corresponding machine idle-time or over-time. Idle-time is normally avoided by maintenance of high in process inventories; however, this practice also serves to mask the existence of schedule imbalance. Deane [10] recognized the possibility of reducing in-process inventories by using a dispatching methodology to balance machine workloads over a multi-period planning horizon. This results from the increased ability to plan a machine's work time as well as its down time thus creating a minimum of unanticipated idle time.

Balancing of machine workloads can also be related to other job-related measures of performance. In the context of this research, balance implies that machine workload completion times are clustered about the average machine workloading (mean flow time). Thus, one degree of imbalance can be measured by the makespan time - the maximum workloading of any machine. Minimum makespan time would, then, be represented by perfect balance.

The satisfaction of due dates is consistently stressed in the literature as an important measure of any scheduling procedure. Ashour [1] states that, "the ability to meet pre-assigned due dates undoubtedly dominates other criteria," in importance. There is no reason to believe

that the use of a balance-oriented scheduling procedure would result in an increase in the frequency of job lateness. On the contrary, the predictability associated with the periodic scheduling procedure necessitated by the definition of balance used in this research may serve to decrease the job lateness rate.

Problem Description

This research deals with the problem of scheduling M batch-type jobs on N parallel processors with the intent of examining optimal and heuristic methods of accomplishing this scheduling under the criteria of minimizing total change-over cost while maintaining a "balanced" workload over all machines. The set of M jobs have sequence-dependent change-over costs but are otherwise independent; that is, there are no precedence or technological restrictions over the set of all possible sequences. The unconstrained minimization problem treated herein becomes constrained only when workload balancing restrictions are imposed.

Sequence dependent change-over cost refers to the cost associated with the work required to make ready a machine, $n \in N$ presently processing job i to process job j such that $i, j \in M$. This cost denoted $\tilde{c}_{ijn}$ is assumed to be deterministic and typically displayed in array form. If all parallel processors are identical then $\hat{c}_{ijn} = \hat{c}_{iju}$ for all $i, j \in M$ and for all $n, u \in N$; and the array $\hat{C}$ is two dimensional. If, however, as in the case of this research, $\hat{c}_{ijn} \neq \hat{c}_{iju}$ for some $i, j \in M$ and for some $n, u \in N$ then $\hat{C}$ is a three dimensional array and the problem is said to involve distinct processors.

Following Marsh [12] the M x M matrices of real job change-over costs (one for each machine) are each imbedded in a larger (M + N) x (M + N) matrix of change-over costs in which jobs M + 1 through M + N are artificial jobs representing initial and final states (possibly idle). It is assumed that there is a cost associated with 'start-up' from the initial state and 'shut-down' to the final state and that these costs may vary among the machines. Thus, if M jobs are to be processed on N machines, the cost of change-over from the initial state of machine $n \in N$ to the first real job assigned to machine n will be defined as $c_{M+n,j,n}$ and cost associated with change-over from the final real job completed on machine n to the final state of machine n will be defined as $c_{i,M+N+n,n}$. An array, so constructed, is shown in (I-1).

$$\hat{C}_n = \begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ M \\ M+1 \\ \vdots \\ M+N \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & . & . & . & M & M+1 & . \quad . \quad . \quad M+N \end{array} \\ \left[\begin{array}{cc|c} & & \\ \multicolumn{2}{c|}{E} & B \\ & & \\ \hline & & \\ \multicolumn{2}{c|}{A} & D \\ & & \end{array}\right] \end{array} \qquad \text{(I-1)}$$

Submatrix E contains the original change-over costs for the set of M jobs on machine n. Row M+n in the submatrix A contains the start-up costs for this machine. All remaining rows in this submatrix represent start-up costs for the remaining N-1 machines and, therefore,

must be prohibited. This is done by assigning an infinite cost to these elements. In like manner, the (M+n) column of submatrix B represents the change-over costs associated with shut-down on this machine. Remaining elements in submatrix B are again assigned infinite values to prohibit shut-down on a machine other than the one assigned to process the final job. Submatrix D is an (NxN) matrix in which all diagonal elements are zero and all off-diagonal elements are infinite.

Sequentially infeasible change-overs are also precluded by assigning to them an infinite change-over cost. For example, the change-over cost $c_{i,i,n} = \infty$ due to the assumed non-partitive nature of the jobs allowing job i to be processed only once.

A sequence S_n on machine n is represented by a vector of m' ordered pairs of jobs,

$$S_n = [(M+n, i_{1,n}), (i_{1,n}, i_{2,n}), \ldots, (i_{m',n}, M+N+n)] \qquad (I-2)$$

where $i_{j,n}$ denotes the $(j+1)^{st}$ job on machine n and $m' < M$. Furthermore, associated with any feasible schedule S_n is a balancing criteria which requires that the completion time of the (m')th job of S_n for all $n \varepsilon N$ be within an absolute time interval of the average machine work-loading. In this research the average machine loading is also the minimum makespan time.

Similarities with Known Problems

If this problem is considered without the balancing restrictions and the processors are considered identical, the problem becomes an augmented traveling salesman problem. A solution procedure to this

problem involving dependent change-over times has been treated previously [12]. In the augmented problem the change-over costs are arrayed in a matrix similar to that of (I-1) except that all the rows in submatrix A are equal as are all the columns of submatrix B. The key to this solution procedure lies in the fact that the optimal solution to the M city problem is imbedded in the solution set of the M+N city problem.

As Marsh [12] points out, when distinct processors are considered, the problem can no longer be handled in a straightforward manner in which the augmented traveling salesman problem can be treated; however, the extension is not a major one.

CHAPTER II

LITERATURE SURVEY

This research covers three distinct areas of general scheduling theory; consequently, the literature survey will be presented in three sections: (1) research concerning the branch and bound algorithm used in this research, (2) previous work done in the area of workload balancing, and (3) heuristic methods used for scheduling under various alternate assumptions and criteria.

The Branch and Bound Algorithm

A branch and bound algorithm for determining the optimal sequence of a set of M jobs on a single processor was first presented by Little et al. [14]. Research done by Pierce and Hatfield [13] extended the algorithm of Little et al. to include the meeting of preassigned due dates while minimizing change-over times. This research led to suggestions for extending their algorithm to the case of parallel processors. These suggestions were used by Marsh [12] to develop a branch and bound algorithm for scheduling on either distinct or parallel processors.

Marsh's algorithm is designed to minimize sequence-dependent change-over times while meeting predetermined due date restrictions. All jobs are assumed available at some arbitrary time zero and to be sequence independent with respect to processing times. No job splitting is allowed. The algorithm used in this study to obtain optimum balanced

solutions is an adaptation of that developed in [12].

Workload Balancing

Research in the area of workload balancing has been performed under various disciplines and for various types of sequencing problems. Balancing, as it applies to this research, does not have the usual connotation of assembly line balancing but, rather, is the equalization of parallel processor workloads.

Eilon and Chritofides [5] use a zero-one programming method and a heuristic method for solving the problem of allocating n objects of given magnitude Q_i to boxes, each box having a capacity C_j, in such a way that the capacity constraints are not violated and the number of boxes required is a minimum. One of the alternate optimality criteria mentioned, but not used, in their research is that of minimizing unused capacity. Greenberg [11] altered the algorithm of Eilon and Christofides to solve the loading problem when the number of machines (boxes) is specified and each machine has an existing workload assigned at time zero. His algorithm is equivalent to placing the largest remaining job on the machine with the least amount of remaining space that will accept it. A modification of this algorithm, allowing equal capacities and providing an initializing step, is used in this research.

Deane and Moodie [10] developed a flow controlled dispatching methodology for balancing machine workload assignments in a dynamic job shop environment. The workload balancing indices which are used are machine oriented, not time period oriented, that is, they are designed to maintain a relatively constant workload on individual machines over

several time periods, not to maintain even distribution over all machines for a given time period.

Irastorza [16] has developed several workload balancing measures for the general job-shop. Most of these measures are also related to multi-time period analysis and are not relevant when analysis is single-time-period oriented.

Heuristic Experience

Gere [8] used simulation to study the effectiveness of heuristic scheduling rules combined with priority rules in the M x N flow shop problem. Jobs consisted of technologically ordered operations and no back passing was allowed. Both the static and dynamic cases were examined with the intent of meeting due dates or, failing this, to minimize the sum of lateness times. Although the program was designed for this due date oriented problem, Gere observed that little effort was required to adapt it to the problem of minimizing makespan time. Limited experience with this adapted program indicated that heuristics may be very effective in handling this problem.

Gavett [6] examined three elementary heuristic rules for sequencing a set of M jobs through a single facility to minimize change-over time. In addition to sequence dependent set-up times, he assumed that the final job in job set M was the first to be processed in the next job set N i.e., no "start-up" or "shut-down" cost was assumed. Gavett's NB or "Next Best" rule made job selections based upon the next least change-over time - much the same as the algorithm of Lockett and Muhlmann [15]. His second rule (NB') is a variation of the NB rule.

It generates M-1 sequences by varying the initial job M-1 times and thereafter following the NB rule. The NB" rule involves the column reduction of the change-over time matrix as defined by Little et al. [12]. After column reduction, the NB" schedule is obtained by applying the NB' rule to the resulting matrix. The study concluded that the NB rule and its variants were significantly better (8% to 76%)than random scheduling.

A more sophisticated heuristic approach to the traveling-salesman problem is presented by Ashour, Vega, and Parker [2]. In this case the objective is to minimize change-over cost. Their method employs the maximum alternate cost criteria to motivate change over selections in a branch and bound heuristic algorithm similar to that of Little et al. and incorporates a look-ahead tie breaking scheme. Of fifteen known problems (selected from the literature) this heuristic provided the optimal solution for eight.

CHAPTER III

MEASURES OF PERFORMANCE

The fundamental assertion of this research is that balancing of workloads on parallel processors is desirable. In order for this general criteria of balancing to be converted to an analytical measure, it is necessary to explain exactly what is meant by the term balance as it relates to the current shop situation. For example, balancing of workloads in this research is single time period oriented; that is, the objective is even distribution of workloads over all machines in a single time period. This places emphasis upon the predictability of machine productivity and overall shop loading. Balance, however, can also be machine oriented as in the case of the measures developed by Irastroza [16]. In this situation emphasis is placed upon the consistency of machine utilization in terms of average workload per time period over a multi-period planning horizon.

In actual practice the decision making authority within the firm will be required to furnish guidelines reflecting the utility of various production measures within the particular processing environment of concern, so that those in charge of scheduling can decide which one of a number of possible balance measures to use. These guidelines will, in general, reflect operating, maintenance, and wage constraints. For purposes of this research some basic measures will simply be defined with the intent of using them to compare the performance of the given optimal and heuristic scheduling procedures when constrained by a

particular balance criterion.

Before continuing, more should be said concerning the single time period orientation of this research. This orientation is based upon the assumption, common in much of the scheduling literature, of a static situation. This assumes that all jobs are available for scheduling at some arbitrary time zero. In this case the assumption is not unrealistic in the sense that continuous arrival of orders is not prohibited. Jobs are simply queued off the job floor. During the processing period, incoming jobs are engineered and, when the desired amount of work is accumulated, jobs are scheduled, as a lot, to be dispatched at the beginning of the next period.

It is important to note that balance, in this research, was considered a constraint rather than a measure to be minimized. The objective was to find the schedule yielding the minimum total changeover cost which also satisfied the fixed constraint. The balance constraint, to be defined, was basically a fixed deviation from perfect balance. The assigned workloading for a given machine was defined as the sum of the processing times of the individual jobs (t_i) assigned to it. Thus if the job set M is to be scheduled on the machine set N, each job to be performed only once, the machine workloading T_n was defined to be:

$$T_n = \sum_i t_i ; \text{ for all } n \in N, \; i \in m' \qquad \text{(III-1)}$$

where $m' < M$ is the subset of M jobs assigned to machine n. The average or optimally balanced workload can then be defined as

$$T^* = \frac{1}{N} \sum_{n=1}^{N} T_n \qquad \text{(III-2)}$$

Balance will then be defined in terms of an absolute time deviation, D, about T^*. A schedule will be said to be balanced if, for all $n \in N$, the workloading of machine n is within the limits imposed by the fixed quantity D, such that,

$$|T_n - T^*| \leq D \quad \text{for all } n \in N \qquad \text{(III-3)}$$

This definition of balance accounts for a certain degree of scheduling flexibility. An amount of processing time, D, in excess of the average workloading (minimum makespan time) is available on each machine. This additional time allows for unforeseen work slowdowns and stoppages as well as a limited amount of rescheduling. This flexibility should aid in lowering the incidence of job lateness.

An additional consequence of this definition is the existence of a large feasible solution space. Within the limits defined by the variable D, there will be many schedules which will meet the balance constraint; consequently, some measures are needed which will allow comparisons to be made between the various proposed scheduling procedures.

Dispersion Indices

In the context of single period balancing, the most general measure of balance can be defined in terms of the absolute deviation of the machine workloadings about T^*. The first measure, Maximum Dispersion Index (MDI), determines the greatest absolute deviation from T^* of the individual machine schedules, S_n. Thus

$$MDI = \max_{N} \frac{|T_n - T^*|}{T^*} . \qquad \text{(III-4)}$$

The Average Deviation Index (ADI) measures the average deviation of assigned schedules about T^* without regard to cancelling effects of over- and under-utilization of machines within the time period. It is defined as,

$$ADI = \frac{1}{N} \sum_{n=1}^{N} \frac{|T_n - T^*|}{T^*} \qquad \text{(III-5)}$$

Overtime Index

This index provides a measure of the relative amount of machine over-utilization within a scheduling period. It measures the total processing time in excess of T^* required by any complete schedule generated by the scheduling procedures. It is defined as:

$$OTI = \sum_{n=1}^{N'} (T_n - T^*) \qquad \text{(III-6)}$$

where N' is the subset of N for which the assigned processing times are greater than T^*.

This measure is important when, for reasons attributable to increased wages or to the possible backlog of jobs into succeeding periods, a high cost is associated with workloading beyond T^*.

Extended Limit Index

Consider the case where balancing limits have been imposed. If

all machine workloadings are within these limits, then the generated schedule is considered balanced. Depending upon the size of D and the variance of the job processing times, it may not be possible to obtain a feasible schedule using heuristic rules which do not possess a look-ahead scheme. It then becomes necessary to have a measure of the frequency of the occurrance of infeasible schedules.

These indices, then, measure the amount of processing time beyond $(T^* + D)$ and short of $(T^* - D)$ resulting from the scheduling procedures. If we let $R = T^* + D$ be the upper balance limit and $-R = T^* - D$ be the lower balance limit then the absolute extended limit time interval becomes

$$L_n = \begin{cases} T_n - R ; & \text{if } T_n > R \\ R - T_n ; & \text{if } T_n < -R \\ 0 ; & \text{otherwise} \end{cases}$$

It is now possible to define the Maximum Extended Limit Index as,

$$MEL = \max_N \frac{L_n}{T^*} \qquad \text{(III-7)}$$

and the Average Extended Limit Index as,

$$AEL = \frac{1}{N} \sum_{n=1} L_n / T^* \qquad \text{(III-8)}$$

These indices are good measures of a procedure's performance when a high cost is associated with infeasible schedules but cost variance within the feasible range is minimal or acceptable.

CHAPTER IV

BRANCH AND BOUND ALGORITHM

This algorithm is an adaptation of the branch and bound algorithm of Marsh [12]. It is designed to minimize total change-over cost of the schedule of M batch type jobs on N parallel but distinct processors such that the total workload on any processor meets an imposed balancing restriction. Distinct processors imply that change-over costs are not only sequence dependent but also processor dependent. It seems reasonable to assume that, in any manufacturing environment, a certain amount of machine replacement will be continually on-going due to technological obsolescence. At the same time, economic constraints will be acting to limit replacement during any given budget year resulting in a mixture of technologically dissimilar processors in use at a given time. It is logical, then, that the change-over costs will in general, vary between processors. For this very practical reason this research has been conducted under the assumption of distinct processors.

Branching

The process of branching consists of breaking the total set of possible schedules into successive dichotomous subsets which include or exclude particular job change-overs. Each branching point is called a node. The first node of the tree represents the set of all possible

schedules (Q). This node is partitioned into Q_i' representing all schedules which include the change-over (p,q) on processor n, and Q_i'' representing all schedules which exclude the selected change-over $(p,q)_n$. Thus

$$Q_i' = S \mid S \in Q \; ; \; (p,q)_n \in S \tag{IV-1}$$

$$Q_i'' = S \mid S \in Q \; ; \; (p,q)_n \notin S \tag{IV-2}$$

The selection procedure for the specific change-over $(p,q)_n$ is motivated by the maximum alternate cost concept described by Little et al. [14] and extended to the parallel processor problem by Marsh [12]. If a change-over $(p,q)_n$ is not made on processor n, then exactly one of two events must occur: (1) the change-over (p,q) is not made on any processor or (2) the change-over (p,q) is made on a processor other than n. If (1) occurs then change-overs $(p,u)_r$ $u \neq q$ and $(v,q)_t$ $v \neq p$ must be made. The occurrence of (1) or (2) is necessary because each job must be processed.

In case (1) the alternate cost of any schedule is:

$$\gamma_{pgn} = \min_{\substack{1 \le r \le N \\ u \neq q}} \hat{c}_{pur} + \min_{\substack{1 \le r \le N \\ v \neq p}} \hat{c}_{vqr} \tag{IV-3}$$

because change-overs from p and to q must occur somewhere in the schedule other than on processor n.

In case (2) the alternate cost of any schedule will be:

$$\xi_{pgn} = \min_{r \neq n} \hat{c}_{pgr} \tag{IV-4}$$

Lower Bounds

Little et al. [14] define a process of subtracting the smallest element in any row or column of a matrix from each of the corresponding row and column elements as row (column) reduction with the resulting sum of the minimum elements being a lower bound on any schedule. Following Marsh [12] the three dimensional array $\hat{C}$ described in (I-1) is said to be fully reduced if for each i there exists some h and k such that $\hat{c}_{ihk} = 0$ and for each j there exists some f and g such that $\hat{c}_{fjg} = 0$. Thus

$$\hat{c}'_{ijn} = \hat{c}_{ijn} - \min_{v,r} \hat{c}_{ivr} - \min_{u,r} [\hat{c}_{ujr} - \min_{v,r} \hat{c}_{uvr}] \qquad \text{(IV-5)}$$

It can be shown that

$$h = \sum_i \min_{v,r} \hat{c}_{ivr} + \sum_j \min_{u,r} [\hat{c}_{ujr} - \min_{v,r} \hat{c}_{uvr}] \qquad \text{(IV-6)}$$

is a lower bound on the total change-over time for any schedule under $\hat{C}$.

If a composite matrix constructed in this manner:

$$C^* = c^*_{ij} = \min_r \left\{ \hat{c}_{ijr} \right\} \qquad \text{(IV-7)}$$

is reduced in the manner described above the result is:

$$c^{*'}_{ij} = c^*_{ij} - \min_v c^*_{iv} - \min_u [c^*_{uj} - \min_v c^*_{uv}] \qquad \text{(IV-8)}$$

and a lower bound on any schedule which is an element of T_i is

$$h = \sum_i \min_v c^*_{iv} + \sum_j \min_u [c^*_{uj} - \min_v c^*_{uv}] \qquad \text{(IV-9)}$$

Furthermore, bounding from below the cost required for any subset of $Q_i' \in Q$ defined by (IV-1) can be expressed in terms of this lower bound;

$$b(Q_i') = b(Q_i) + h \tag{IV-10}$$

The lower bound of any schedule that is an element of Q_i'' defined by (IV-2) can be expressed in terms of the alternate costs;

$$\theta_{ijn} = \min [\gamma_{ijn}, \xi_{ijn}] \tag{IV-11}$$

and it is proved in Marsh [12] that the cost $z_o(S)$ of any admissible schedule S such that $(i,j)_n \notin S$ is bounded as follows:

$$z_o(S) \geq \min [\gamma_{ijn}; \xi_{ijn}] = \theta_{ijn} \tag{IV-12}$$

Therefore the bound on any $S \in Q_i''$ is expressed as:

$$b(Q_i'') = b(Q_i) + \theta_{pqn} \tag{IV-13}$$

Thus if we are presently at node Y of the tree, a lower bound on the next inclusive node X would be

$$b(X) = b(Y) + h$$

and a lower bound on the exclusive node $\bar{X}$ would be

$$b(\bar{X}) = b(Y) + \theta_{pqn}$$

for the change-over $(p,q)_n$.

Branching always proceeds to the inclusive node until a complete

solution is obtained or until a selection is made which is infeasible due to the balancing restriction. If a complete feasible schedule is obtained and there are no nodes which have a lower bound less than the current least cost then the solution is optimal. If, however, there exists a node with a lower bound less than the present cost we must backtrack to that node and proceed along the path of the exclusive node until a new solution is reached or until the cost of the last complete solution is exceeded.

If any assigned change-over results in a violation of the balance constraint, the assignment is rejected and another change-over must be selected. This amounts to branching along the path of the exclusive node.

Of these two conditions that may require backtracking the latter warrents further explanation. If backtracking is necessitated by a violation of the balance restriction prior to reaching a complete solution, the resulting initial solution will necessarily possess a lower bound which is equal to or greater than that of the initial solution of the unconstrained problem. This results from the requirement to choose a path other than the all inclusive path. If the resulting lower bound is greater, and it typically is, the solution set of schedules with lower bounds equal to or less than the current one is also larger. Guaranteeing optimality is tied to a set partitioning scheme which systematically searches all exclusive nodes with lower bounds less than the lower bound of the current complete solution. Replacement of the current best solution occurs when a lesser lower bound is obtained. Because of this procedure, any increase of the lower bound of the initial

unconstrained optimal solution results in an increase in the size of the remaining solution set and thus decreased efficiency of the algorithm.

Statement of the Algorithm

Step 1: Construct C according to (I-1) and C* according to (IV-7). Let z_o be the total change-over cost for the best schedule available at any time in the algorithm, initially ∞.

Step 2: Row and column reduce C according to (IV-5) and C* according to (IV-8). Compute h from (IV-9) and let b(Q) = h.

Step 3: Compute the alternate costs θ_{ijn} according to (IV-11) for each change-over (i, j) for which $c^*_{ij} = 0$. Let $\theta_{pqn} = \max\ \theta_{ijn}$.

Step 4: Branch from the current node, say Y, by creating a node X for schedules including change-over (p,q) and a node $\bar{X}$ for schedules which prohibit change-over (p,q), i.e. $(\overline{p,q})$. Compute the lower bound $b(\bar{X})$ according to (IV-13).

Step 5: Develop the data describing the problem at node $\bar{X}$ by letting $\hat{c}_{pgn} = \infty$ and $c^*_{pq} = \min_u\ \hat{c}_{pqu}$.

Step 6: Develop the penalty cost matrices for the new scheduling problem at node X by the following procedure:

(a) Delete rows p and columns q from $\hat{C}$ for all k = 1,...,N.

(b) Find the starting job s and the final job e in the partial sequence containing $(p,q)_n$ and set $\hat{c}_{sen} = \infty$. Also set $\hat{c}_{qpn} = \infty$ for all n = 1,...,N

(c) If (p,q) is imbedded in a partial sequence containing terminal job M+N+n set $\hat{c}_{M+y,p,y} = \infty$ for all $y \neq n$.

(d) If job e is the ending job in the partial sequence containing M+n, and, if for any other processor, $u \neq n$, job s is the starting job in the partial schedule containing $M+N+u$ set $c_{esn} = \infty$.

Recompute C^* according to (IV-7).

Step 7: Row and column reduce C^* according to (IV-8) and C according to (IV-5). Compute h from (IV-6) and b(X) from (IV-10).

Step 8: If $b(X) \leq z_o$ and the change-over does not create a partial sequence S_n which assigns more than $\bar{T} + D$ units of work to machine n, thus violating the balance criteria, go to Step 9. If $b(X) > z_o$ or a partial sequence is unbalanced, backtrack to the node W representing the greatest number of feasible change-overs. Recall the cost matrics for the partial schedule represented by node W and go to step 3 setting Y = W. If no such node W exists terminate the solution procedure. The last schedule found is optimal.

Step 9: If the reduced matrix C^* at node X has more than two rows and two columns which are not all infinite go to step 3. If not, determine which, if any, job is not included in the partial schedule. If the remaining change-overs result in complete sequence S_n on machine n such that $\sum_{i \epsilon n} t_i > \bar{T} + D$ or such that $\sum_{i \epsilon n} t_i < \bar{T} - D$ for any $n \epsilon N$ then the partial schedule is infeasible. In this case backtrack to the node W representing the greatest number of feasible change-overs. Recall the cost matrices for the partial schedule represented by node W and

go to step 3 setting Y = W. If the partial schedule satisfies the feasibility checks, complete the schedule by adding those change-overs which have one zero in each non-infinite row and column of $C^{*'}$. If the lower bound of this complete solution is greater than z_0 backtrack according to step 8.

Numerical Example

To illustrate the operation of the algorithm consider the M = 5, N = 2 distinct parallel processor problem whose change-over cost matrices are shown in (IV-14) and (IV-15)

$$\hat{C}_1 = \begin{bmatrix} \infty & 8 & 4 & 7 & 2 & 4 & \infty \\ 2 & \infty & 7 & 2 & 4 & 7 & \infty \\ 2 & 5 & \infty & 6 & 3 & 5 & \infty \\ 4 & 4 & 3 & \infty & 5 & 4 & \infty \\ 1 & 2 & 3 & 1 & \infty & 6 & \infty \\ 6 & 5 & 5 & 3 & 5 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix} \qquad \text{(IV-14)}$$

$$\hat{C}_2 = \begin{bmatrix} \infty & 8 & 4 & 7 & 2 & \infty & 4 \\ 2 & \infty & 7 & 2 & 4 & \infty & 5 \\ 2 & 5 & \infty & 6 & 3 & \infty & 5 \\ 4 & 4 & 3 & \infty & 5 & \infty & 3 \\ 1 & 2 & 3 & 1 & \infty & \infty & 6 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 6 & 5 & 4 & 6 & 5 & \infty & \infty \end{bmatrix} \qquad \text{(IV-15)}$$

Let the processing times for the given jobs be $P = \{5,4,6,3,8\}$. In line with previous definitions jobs 6 and 7 are initial jobs and

jobs 8 and 9 are final jobs (imaginary). From step 1 the penalty cost matrix for each machine $\hat{C}_1$ and $\hat{C}_2$ are constructed as shown above. Construct C^* according to (IV-7) as shown below.

$$C^* = \begin{bmatrix} \infty & 8 & 4 & 7 & 2 & 4 & 4 \\ 2 & \infty & 7 & 2 & 4 & 7 & 5 \\ 2 & 5 & \infty & 6 & 3 & 5 & 5 \\ 4 & 4 & 3 & \infty & 5 & 4 & 3 \\ 1 & 2 & 3 & 1 & \infty & 6 & 6 \\ 6 & 5 & 5 & 3 & 5 & \infty & \infty \\ 6 & 5 & 4 & 6 & 5 & \infty & \infty \end{bmatrix} \qquad \text{(IV-16)}$$

From step 2 C^* and $\hat{C}$ are reduced to yield

$$\hat{C}_1' = \begin{bmatrix} \infty & 5 & 1 & 5 & 0 & 1 & \infty \\ 0 & \infty & 4 & 0 & 2 & 4 & \infty \\ 0 & 2 & \infty & 4 & 1 & 2 & \infty \\ 2 & 1 & 0 & \infty & 3 & 0 & \infty \\ 0 & 0 & 1 & 0 & \infty & 4 & \infty \\ 3 & 1 & 1 & 0 & 2 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix} \qquad \text{(IV-17)}$$

$$\hat{C}_2' = \begin{bmatrix} \infty & 5 & 2 & 5 & 0 & \infty & 0 \\ 0 & \infty & 5 & 0 & 2 & \infty & 1 \\ 0 & 2 & \infty & 4 & 1 & \infty & 1 \\ 1 & 0 & 0 & \infty & 2 & \infty & 0 \\ 0 & 0 & 2 & 0 & \infty & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & 0 & 2 & 1 & \infty & \infty \end{bmatrix} \qquad \text{(IV-18)}$$

$$C^{*'} = \begin{bmatrix} \infty & 5 & 2 & 5 & 0 & 1 & 2 \\ 0 & \infty & 5 & 0 & 2 & 4 & 3 \\ 0 & 2 & \infty & 4 & 1 & 2 & 3 \\ 1 & 0 & 0 & \infty & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 & \infty & 4 & 5 \\ 3 & 1 & 2 & 0 & 2 & \infty & \infty \\ 2 & 0 & 0 & 2 & 1 & \infty & \infty \end{bmatrix} \qquad \text{(IV-19)}$$

A lower bound on the total changeover cost required by any feasible schedule is h = 19 and the tree of Figure 1 is started.

According to step 3, element (4,9) of $C^{*'}$ has the maximum alternate cost of 2; therefore, $(p,q)_n = (4,9)_2$ since $c^*_{4,9} = \hat{c}_{4,9,2}$.

By step 4 the set Q thus far is partitioned into Q_1' denoted by $(4,9)_2$ in Figure 1 and Q_1'' denoted by $(\overline{4,9})_2$.

The cost data to be computed in step 5 describing the scheduling problem subset Q_1'' is identical to equations (IV-17), (IV-18) and (IV-19) except that $c^*_{6,4} = \infty$ and $\hat{c}_{6,4,1} = \infty$.

All changeovers both to and from job 4 must now be done on machine number 2. The adjustments of step 6 yield:

$$\hat{C}_1 = \begin{bmatrix} \infty & 5 & 1 & \infty & 0 & 1 & \infty \\ 0 & \infty & 4 & \infty & 2 & 4 & \infty \\ 0 & 2 & \infty & \infty & 1 & 2 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & 1 & \infty & \infty & 4 & \infty \\ 3 & 1 & 1 & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix} \qquad \text{(IV-20)}$$

$$\hat{C}_2 = \begin{bmatrix} \infty & 5 & 2 & 5 & 0 & \infty & \infty \\ 0 & \infty & 5 & 0 & 2 & \infty & \infty \\ 0 & 2 & \infty & 4 & 1 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & 2 & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 2 & 0 & 0 & 2 & 1 & \infty & \infty \end{bmatrix} \qquad \text{(IV-21)}$$

$$C^* = \begin{bmatrix} \infty & 5 & 2 & 5 & 0 & 1 & \infty \\ 0 & \infty & 5 & 0 & 2 & 4 & \infty \\ 0 & 2 & \infty & 4 & 1 & 2 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & 2 & 0 & \infty & 4 & \infty \\ 3 & 1 & 2 & \infty & 2 & \infty & \infty \\ 2 & 0 & 0 & 2 & 1 & \infty & \infty \end{bmatrix} \qquad \text{(IV-22)}$$

The reduction of C^* in equation (IV-22) required by step 7 results in the reduction of row 6 by 1 and of column 6 by 1. Therefore $b(X) = 19 + 2 = 21$ as shown in Figure 1. Step 8 deduces that there are more than two remaining rows and columns with non-infinite elements and that $b(X) = 20 < z_o = \infty$.

Recursive iterations of this procedure are required and will result in the tree representation in Figure 1. A tree representation of the unconstrained solution to this problem is also given in Figure 1 (dotted lines). Notice that increased backtracking is necessary in the constrained problem due to the fact that the balancing restriction forced the algorithm off the 'all-inclusive' path before a complete initial solution was obtained, increasing both the initial solution cost (z_o), and the size of the remaining solution space.

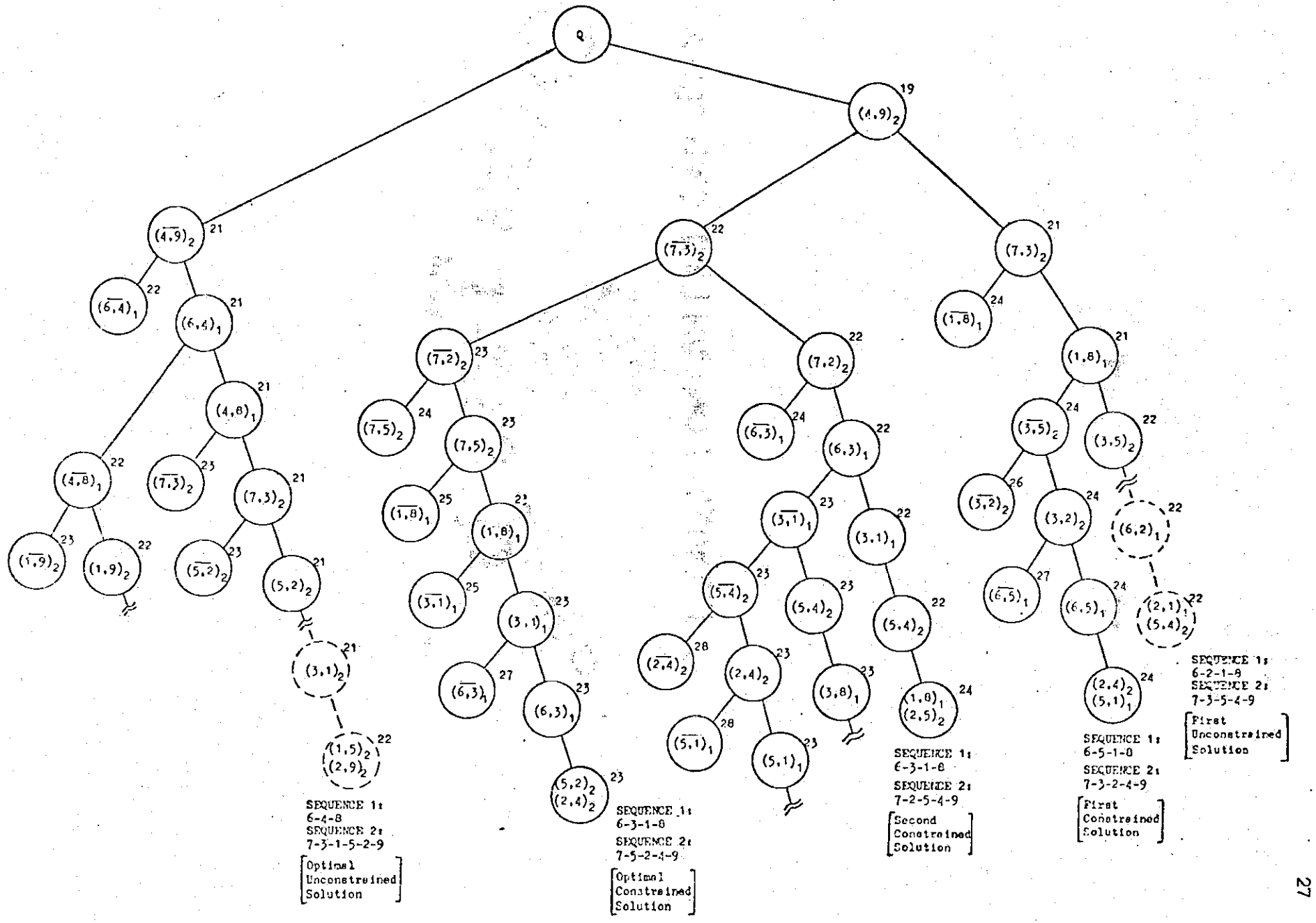

Figure 1. Tree Representation of Distinct Parallel Processor Problem where M = 5, N = 2, D = .15

CHAPTER V

HEURISTIC RULES

General

In normal operating situations two factors contribute to the practical difficulty of obtaining optimal sequences for parallel processors. The first involves the $\hat{c}_{ijn}$ values representing job change-over costs. The problem of obtaining these costs is not trivial especially when the change-over job associated with the cost is often complex and not amenable to exact standardization. For analytical purposes this problem is minimized by assuming that change-over costs are deterministic. The second difficulty is not so easily handled. It relates to the availability of computing facilities - a luxury which in many cases might not exist. Furthermore, as problems become even moderately large, existing optimal scheduling procedures are limited by the capacity of existing facilities as well as imposed budgetary constraints. It is this factor, then, that necessitates the use of non-optimal, simplified decision rules in the scheduling process. Six such rules are examined in this study.

Although this specific problem has not been addressed in the open literature, researchers concerned with structurally similar problems have employed job selection criteria that could be modified to meet present assumptions. Of specific interest were selection procedures based upon change-over cost used to solve the traditional traveling

salesman problem as well as procedures based upon processing times employed by Eilon and Christofides [5] and Greenberg [11] in their research of the loading problem.

Maximum Regret Algorithm

This algorithm with a tie breaking scheme was used by Ashour, Vega, and Parker [2] in solving the traveling salesman problem as it applies to single processor scheduling. In order to be used in the parallel processor problem with balancing restrictions it must be modified to incorporate a back-tracking scheme to be used when the constraining criteria forbid a particular change-over. It is, in effect, a first pass solution of the optimal algorithm described previously. There is no provision, however, for breaking ties between the maximum alternate costs at any point in the algorithm. Instead, if a tie exists, it is broken by a random selection of the alternative choices. Thus the statement of the algorithm is the same as that presented in Chapter IV except that in step 8, when the first feasible schedule is obtained, the procedure is terminated.

Least Cost Next Rule

The rule (LCN) is an extension of that used by Lockett and Muhlemann [15] and Gavett [6] in their research of the traveling salesman problem. Under this rule (LCN), a job is selected which corresponds to the minimum change-over cost over all N machines. Thus, initialization occurs by determining the least start-up cost for each machine and selecting the job corresponding to the minimum of these

costs. Minimum change-over costs on the N-1 machines not selected remain the same for the next selection while the minimum change-over cost of the previously selected machine must be made from the row of the change-over cost matrix corresponding to the previously selected job. The procedure continues until a selected job violates the balance constraint, in which case, the job corresponding to the next cheapest and feasible change-over cost is selected.

Statement of the Rule

(1) From the change-over cost matrices, select the minimum feasible change-over (possibly start-up) cost for each machine.

(2) Select the job and machine corresponding to the minimum of these costs.

(3) If $T_k \leq T^* + D$, eliminate the selected job from further consideration and go to step (1). If $T_k > T^* + D$ go to step (4).

(4) Select the next cheapest change-over cost from the remaining machines. Go to step (3).

Least Cost - Least Remaining

This rule (LCLR) is a variant of the previous rule. Instead of assigning jobs based strictly upon change-over costs, this rule is concerned with the placement of selected jobs; that is, instead of allowing jobs to be assigned to machines completely at random, the assignments are made in an ordered fashion. Under this rule, one job is initially assigned to each machine based upon the least start-up cost. Thereafter, selections are made by determining the minimum of the change-over costs on the processor with the least remaining

capacity. If the selected job violates the balance constraint, do not assign it. Instead, choose the machine with the next least remaining capacity and select the minimum change-over cost job from it.

Statement of the Rule

(1) Initialize each processor by assigning to it the job with the least change-over cost.

(2) Select the processor with the least remaining capacity.

(3) From the remaining jobs, select the one that has the least change-over cost on the processor selected in step 2. Add the processing time (t_m) of this job m to the selected processor total, such that $T_k = T_k + t_m$. Eliminate job m from further consideration.

(4) If all jobs have been assigned, terminate the procedure. If $T_k \leq T^* + D$ go to step 2. If $T_k > T^* + D$, go to step 5.

(5) Temporarily eliminate processor k from consideration, free job m, and go to step 2.

Least Cost -Greatest Remaining

This rule (LCGR) is identical to the LCLR rule except that ordering of the selections insures even filling of the set of N machines rather than filling of one machine at a time. Under this rule, the machine with the greatest remaining processing capacity is selected and the job corresponding to the least feasible change-over cost on that machine is assigned.

Statement of the Rule

(1) Initialize each processor by assigning to it the job with the least change-over cost.

(2) Select the processor with the greatest remaining capacity.

(3) From the remaining jobs, select the one that has the least change-over cost on the processor selected in step 2. Add the processing time (t_m) of this job m to the selected processor total, such that $T_k = T_k + t_m$. Eliminate job m from further consideration.

(4) If all jobs have been assigned, terminate the procedure. If $T_k \leq T^* + D$ go to step 2. If $T_k > T^* + D$, go to step 5.

(5) Temporarily eliminate processor k from consideration, free job m, and go to step 2.

Largest Job -Least Cost

This rule, referred to as LJLC, assigns jobs primarily on the basis of job size. It assumes that it is advantageous to assign jobs with the longest processing time first. This should reduce wide variance in possible assignments near the end of the scheduling process. Under this rule, jobs are ordered in decreasing order of size and schedules, in order, on the machine which has the least change-over cost. If a balance constraint is violated the job is assigned to the machine with next least change-over cost. If the job violates the balance constraint on every machine, it is arbitrarily assigned to the machine with the greatest remaining capacity.

Statement of the Rule

(1) List the job processing times t_i in decreasing order of magnitude.

(2) Assign the first ordered (largest) job to the processor on which it has the least start-up cost.

(3) Choose the next job and assign it to the processor on which it has the least change-over or start-up cost. Add the processing time t_i of this job to the present total processing time on processor k to which it has been assigned.

(4) If M jobs have been assigned terminate the solution procedure. If $T_k \leq T^* + D$ go to step 3. If $T_k > T^* + D$ go to step 5.

(5) Assign this job to the processor with the next (Least) change-over cost. Go to step 3.

Largest Job Next #1

The next two rules are extensions of Greenberg's [11] loading algorithm. The original intent of this rule is to produce good balance; consequently, it is not expected that they will provide good results with respect to minimizing change-over cost. After initialization, the remaining jobs are assigned, in decreasing order of processing length to the machine with the least remaining capacity. If the balance constraint is violated, the next largest job is assigned until an assignment is made or all jobs have been tried. In the latter case, the machine with next least processing time remaining is selected and the largest job is again assigned to it.

The purpose in using these algorithms is to obtain an indication of the trade-off involved between balance and total change-over cost.

Statement of the Rule

(1) Assign the largest job to the processor on which it has the least start-up cost.

(2) Select the processor with the least remaining capacity and assign the largest remaining job to it.

(3) If $T_k > T^* + D$, temporarily eliminate this job from further consideration and assign the next largest job to the same processor. If none of the remaining jobs can be assigned to this processor eliminate this processor from future consideration and go to step 2. If all jobs have been assigned terminate the solution procedure.

Largest Job Next #2

The algorithm (LJN2) is a variant of the LJN1 algorithm employing a slightly different initiatizing procedure. Here it was felt that varying the initialization process would yield different results in terms of total change-over cost while maintaining relatively equal balance results as obtained by LJN1.

Statement of the Rule

(1) For initiation of the procedure, assign one job to each processor based upon the least start-up cost for that processor.

(2) Select the processor with the least remaining capacity and assign the largest remaining job to it.

(3) If $T_k > T^* + D$, temporarily eliminate this job from further consideration and assign the next largest job to the same processor. If none of the remaining jobs can be assigned to this processor eliminate this processor from future consideration and go to step 2. If all jobs have been assigned terminate the solution procedure.

CHAPTER VI

RESULTS AND DISCUSSION

General

Analysis was primarily concerned with detection of differences between several heuristic scheduling rules (Chapter V) with respect to various measures of performance (Chapter III). In addition, balanced optimal and unconstrained optimal solutions were obtained for a limited set of problems to be used for comparative purposes. In this way, a measure of the tradeoff, between balance and cost was obtained. Accomplishing this required the branch and bound algorithm of Chapter IV coded in Fortran V for the UNIVAC 1108 as well as a Fortran V program for the given heuristic scheduling rules. The Fortran V code for the algorithm to find the optimal balanced solution is given in Appendix C. This program is a modification of the program given in Marsh [12]. The Fortran V coding for the heuristic rules is given in Appendix D.

Problems were defined in terms of a number of processors, N, a number of jobs, M, and a specified Balance Limit (D). Changeover cost matrices were generated for each machine, in any given problem, using uniformly distributed random numbers between 0 and 10. Processing times for the job set were uniformly distributed between 10 and 18.

The random number seeds used to generate the sample changeover cost matrices were subjected to three tests to insure randomness. The

tests included (i) Runs Above and Below the Mean (ii) Autocorrelation, and (iii) Chi-Square Frequency Test [16].

The design of the testing procedure lent itself to the use of Analysis of Variance Techniques. F-tests were conducted to determine if statistically significant differences existed between the mean values of the given balance measures under the given scheduling rules.

Discussion of Results

It was deemed desirable to obtain the true balanced optimal solution for each test problem in order to provide a base with which to compare the heuristic rules. However, the Fortran V code given by Marsh and modified as shown in Appendix C provided severe limitations on the size of problems that could be solved.

With this in mind several problem sets were solved using the optimal algorithm. The results are shown in Appendix B. In each case five replications were run within each problem set. Thus, there were actually five different problems solved for each combination of jobs and machines. Each of the problems were generated using a different random number seed. One sample set of results is shown in Table 1.

Table 2 represents the ANOVA conducted on the total change-over cost data in Table 1. The F-statistic of 29.8 compared with the critical F-value of 2.59 indicates that there is a significant difference in the means of total change-over cost between the scheduling procedures in this problem set.

Table 1. Optimal, Maximum Regret, and Heuristic Results for Total Change-over Cost and Maximum Deviation for the Problem Set in which M = 10, N = 2, D = 15%

	SCHEDULING PROCEDURES								
	OPT(UNC)	OPT(C)	MAXIMUM REGRET (1st Pass)	LJN1	LJN2	LCLR	LCGR	LJLC	LCN
				Total Change-over Cost					
	10	12	19	36	56	16	22	40	19
	8	8	11	43	54	30	22	62	22
	7	9	13	43	54	18	17	41	29
	14	17	22	68	77	29	31	64	30
	11	11	20	51	65	27	21	45	26
Avg	10.0	11.4	17.0	48.2	61.2	24.0	22.6	50.4	25.2
				Maximum Deviation					
	.412	.058	.044	.132	.147	.000	.015	.088	.015
	.355	.118	.118	.145	.145	.132	.013	.079	.013
	.217	.043	.000	.029	.145	.058	.029	.043	.043
	.239	.119	.045	.149	.149	.090	.015	.119	.045
	.405	.000	.054	.135	.135	.027	.081	.027	.027
Avg	.326	.068	.052	.118	.144	.061	.031	.071	.028

Table 2. ANOVA for Total Change-over Cost Data Obtained for Problem Set in which M = 10, N = 2 and D = .15

SOURCE	SS	d.f	MS	F
RULES	13747.9	8	1718.49	29.82
ERROR	2073.9	36	57.61	
TOTAL	15822.0	44		

The results of Duncan's Test concluded that the average unconstrained optimal cost is significantly less than the average balanced optimal cost even though, in two individual cases, the costs were equal. The average cost differences between the unconstrained and balanced optimal solutions for the three problem sets considered are reflected in Figure 2.

M - N	AVERAGE UNCONSTRAINED OPTIMAL COST	AVERAGE BALANCED OPTIMAL COST	PERCENT DIFFERENCE
10 - 2	10.0	11.4	14%
12 - 2	8.4	11.4	35.7%
6 - 3	11.8	19.2	62.7%

Figure 2. Average Difference in Total Change-over Cost Between Unconstrained and Balanced Optimal Solutions for Three Different Problem Sizes.

The large differences in average total change-over costs between the unconstrained and balanced optimal solutions can, in part, be explained by the presence of 'start-up' and 'shut-down' costs. It was observed that, quite often, the unconstrained optimum solution required the sequencing of all jobs on a single machine. In such a case the total change-over cost is the sum of M + 1 uniformly distributed change-over costs. The balanced optimal solutions, on the other hand, always utilize the entire machine set; consequently, the total cost for balanced optimal solutions is always the sum of M + 2N -1 uniformly distributed change-over costs. The effect of this difference in the number of change-over costs which comprise the total cost of either solution should diminish as the number of jobs increase, provided that N remains relatively constant.

The remaining two F-statistics for total change-over cost also exceeded the critical value; therefore, Duncan's tests were conducted for all sets of data. The results indicated that, although the first pass rule produced the best results relative to the next best heuristic solutions-LCGR and LCN- no statistically significant difference between the three rules was detected.

The same statistical procedure was used to compare the performance of the rules and algorithms with reference to maximum and average deviation. For the cases in which two machines were available, Duncan's tests concluded that the 1st Pass rule did not produce results as low as other heuristic rules, specifically LCLR, LCGR and LCN. When N = 3, the 1st Pass rule did produce the best results for both measures; however, in all cases, differences between 1st Pass, LCN, LCLR, and LCGR

were not significant.

Extended Non-Optimal Results

The next phase of the research dealt with the analysis of performance of the heuristic rules, alone. Results for selected large problems are contained in Appendix A.

Problem sizes of 15, 25, 35 and 45 jobs were each scheduled on 3, 4 and 5 machines. Eight problems of each combination of jobs and machines were randomly generated and executed. The results reported in Appendix A reflect averages of each set of problems. A one-way analysis of variance test was conducted using the data from each set of problems and for four measures of performance-cost, maximum deviation, average deviation and extended limit. In every case in which the computed F-statistic exceeded the critical F-value, at the 5% level of significance, Duncan's tests were conducted to determine which pairs of means were significantly different.

The first measure analyzed was again cost. It was expected that the LCGR, LCLR, and LCN rules would produce the best results with respect to cost. F-statistics ranging from 26.2 to 272.8 were obtained. Additionally, every F-statistic exceeded the critical value and Duncan's tests confirmed the existence of a significant difference in average change-over cost between the least cost rules - LCN, LCLR, and LCGR - and the remaining rules which base job selection upon processing times.

Furthermore, in 10 of 12 problem sets, either the LCLR or LCGR rule was significantly better than the remaining rules including LCN. Thus, it seems that there is evidence to support the claim that the LCLR and

LCGR are the best rules for minimizing cost while maintaining "balanced" schedules.

The same procedure was followed to determine if LCGR and LCLR were better than the other heuristic rules with respect to the quality of the balance which they produce. It is important to note that the balance limit, D, used to constrain the scheduling procedures was fixed at 10% for all of the problem sets. This means, then, that each measure reflects the performance of each of the scheduling rules in attempting to meet the imposed constraint.

Several initial runs were executed using a range of values for the balance constraint, D. The results indicated that the given measures of performance were insensitive to variation of this parameter; therefore, a value of 10% was chosen as representation for the test problems considered. A graphical representation of total change-over cost for three heuristic rules applied to a sample problem with varying values of D is displayed in Figure 3. The remainder of the data is tabulated in Table 9 of Appendix A.

The maximum dispersion of any assigned machine workload from the mean flow time (T^*) in any given problem ranged from 0% to 36% of T^*. F-statistics were exceeded in 9 of the 12 problem sets for this measure. Only when the job set was 15 was a rule other than LCLR or LCGR determined to be significantly best. As job and machine sets increased in size LCLR and LCGR clearly dominated other rules in terms of minimizing maximum dispersion (Table 4).

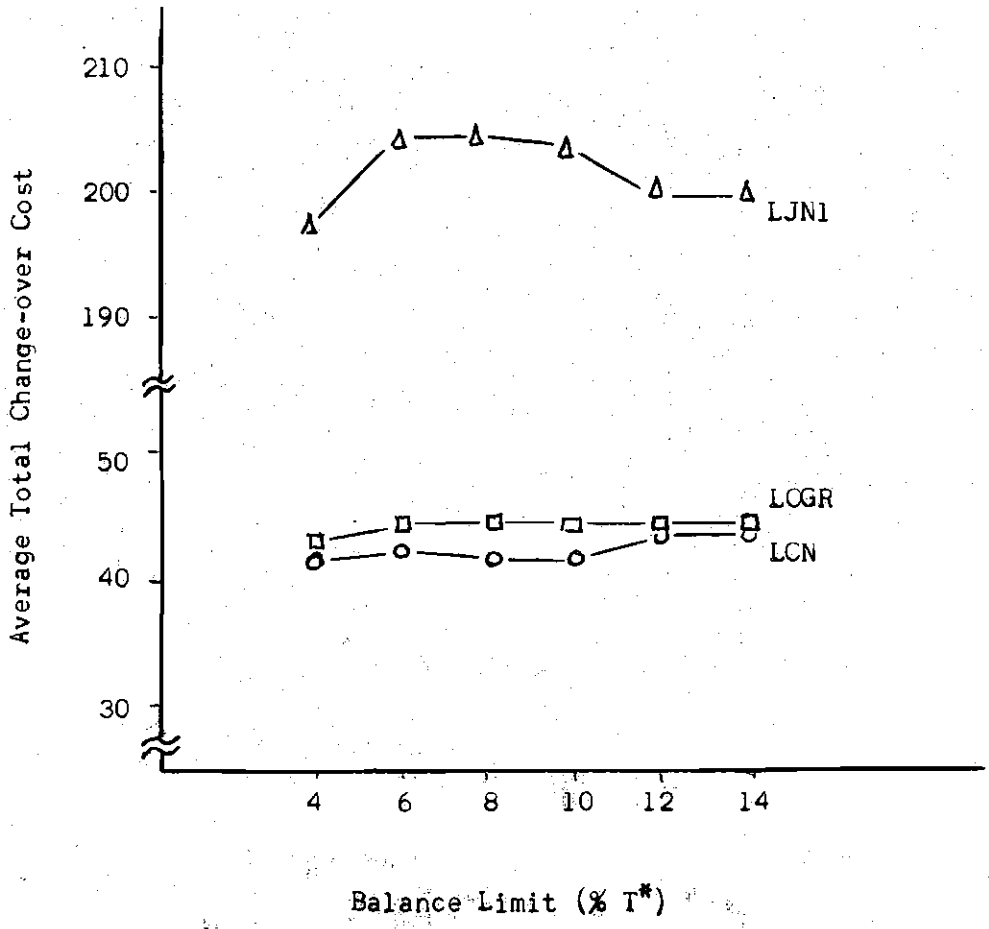

Figure 3. Sensitivity of Average Total Change-over Cost for the Parallel Processor Problem in which M = 40, N = 4 to Varying Values of D.

Results concerning average dispersion are contained in Table 6, of Appendix A. In all but one case, the computed F-statistics exceeded the critical F-values.

Duncan's tests again indicated that when job-lot sizes increase LCLR and LCGR obtain the best results in terms of minimizing average dispersion. Only when M = 15 did rules other than LCLR and LCGR obtain the least average dispersion about T^*. In eight of the nine cases in which Duncan's Tests were performed the LCGR rule produced the least

average dispersion.

Analysis of the data concerning the maximum extended limit measure resulted in conclusions similar to those concerning previous measures. For the smaller job lot size, M = 15, the F-statistic did not exceed the critical value at the 5% level of significance; however, as the number of jobs increased the critical F-value was consistently exceeded. In these cases Duncan's test concluded, in nearly every instance that the LCGR rule produced the significantly best results.

Figure 4 summarizes these results by indicating which rule produced the best average result for each measure and for each problem set. Furthermore it indicates in which problem sets the best rule was determined significant (by Duncan's Test).

Table 4. Indicators of Heuristic Procedures which Produced Best Average Results for the Given Measures of Performance

M	N	HEURISTIC SCHEDULING RULES					
		LCN	LCLR	LCGR	LJLC	LJN1	LJN2
15	3	A*			B*	D	C*
	4	C	A*				B,D
	5		A*		B,C*D		
25	3		A*	B*,C*,D			
	4			A*,B*,C*,D			
	5	A*	B*,C*,D*				
35	3			A*,B*,C*,D*			
	4		A*	B*,C*,D*			
	5			A*,B*,C*,D*			
45	3			A*,B*,C*,D*			
	4		A*	B*,C*,D*			
	5			A*,B*,C*,D*			

A - Total Change-over Cost
B - Maximum Workload Deviation
C - Average Workload Deviation
D - Maximum Extended Limit
(*) - Significantly best (as determined by Duncan's Test)

CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS

Conclusions

In the absence of an efficient optimal algorithm, certain heuristic procedures appear to provide satisfactory results for the problem of scheduling batch-type jobs with sequence-dependent change-over costs on distinct parallel processors when the objective is to minimize total change-over cost and balance workloads. Specifically, the LCGR rule, in addition to producing the best average total cost also produces the best workload balance based on the balance measures used.

A modified branch and bound algorithm provided limited unconstrained and constrained solutions with which to make comparisons and draw conclusions. The following conclusions are based upon these results.

(1) The optimum balanced algorithm produces significantly lower total cost solutions than any of the heuristic procedures examined; however, the LCGR rule produced better balanced solutions.

(2) While the maximum regret (1st Pass) rule produced lower total cost solutions than the LCGR rule, the difference was not significant. Furthermore the balance obtained by the LCGR was again better than that of the 1st Pass rule.

(3) In most cases the balance criteria requires the branch and bound algorithm to take the path of an exclusive node prior to reaching a complete solution. This usually results in an increased lower bound on the initial complete solution and a substantial decrease in the efficiency of the algorithm.

(4) The large difference in average total change-over cost between the unconstrained optimal solutions and the balanced optimal solutions may be exaggerated by the additional change-over costs associated with initial and final states on each processor. Typically, the unconstrained optimal solution involved sequencing all M jobs on a single processor, while the balanced optimal solution required all machines. In this case the total change-over cost for the balanced optimal solution is represented by the sum of 2N - 2 change-over costs more than the M + 1 required by the optimal unconstrained problem. Because each change-over cost matrix is generated using uniformly distributed random numbers, there is no reason to believe that any machine is less expensive to operate. Although not significant in large problems the resulting differences in optimal costs indicate that the effect is significant in small problems.

Exercising the heuristic rules on selected large problems tended to confirm the results obtained from the optimal problem sets. In particular, as the number of jobs increases the LCGR rule consistently

produces the least total change-over cost as well as the best balance.

Recommendations

As a result of the investigation presented in this thesis recommendations for further study can be made.

1. Research aimed at development of a more efficient optimal algorithm for this workload balancing problem is required if problems of significant size are to be analyzed. One approach would be to improve the branch and bound algorithm of this research. This might be accomplished through use of a different bounding technique which would incorporate both balance and cost measures.

2. Realistically, it seems unlikely that the requirement for workload balancing would exist in the absence of other job-related constraints such as pre-assigned due dates or machine limitations. For this reason, an analysis of multi-constrained parallel processor problems in which balance in one constraint is necessary.

3. The only cost treated explicitly in this research is total change-over cost. The function of operations scheduling, however, is concerned with minimizing all the various costs associated with the manufacturing process. These costs are created by factors such as in-process inventory, idle men and equipment, overtime, job-lateness, etc. Several of these factors are, in turn, directly related to and influenced by the scheduling procedure utilized. This research confirms the suspicion that imposing a balance restriction on the scheduling process results in an increase in total change-over cost; thus, if balance is to become an acceptable criteria, it will ultimately

have to be justified on the basis of other associated costs. This results in the need for analysis of the impact of workload balancing upon other functions of operations planning and control such as demand forecasting, inventory planning and control, and dispatching. The continuous and complex nature of the interaction between these functions makes simulation a reasonable approach to such analysis.

4. One of the limitations of the conclusions drawn from this research is due to the factor of problem size. For a more detailed analysis of the effect of this factor on workload balancing, in general, larger problem sizes will have to be considered.

5. Additional relevant balance measures could be developed. Some consideration should be given to weighted measures which reflect not only a degree of balance, but also, total change-over cost. In addition, if balancing, as defined in this research, is analyzed over a multi-period planning horizon, cumulative measures could be used.

6. Because of the fact that none of the tested heuristic procedures was observed to be superior in <u>all</u> cases, it is possible that a procedure representing a composite of the solution rules would produce better solutions. This could be accomplished by development of adaptive decision rules that specify the conditions under which one selection rule should take precedence over another during the scheduling process.

APPENDIX A

HEURISTIC SCHEDULING RESULTS FOR SELECTED LARGE PROBLEM SIZES

Table 5. Sample Means of Total Change-over Cost Results for Selected Large Problems

M	N	HEURISTIC SCHEDULING PROCEDURE					
		LJN1	LJN2	LJLC	LCLR	LCGR	LCN
15	3*	74.38	79.25	51.00	39.75	38.75	32.50
	4*	83.00	88.50	68.88	37.00	37.50	43.00
	5*	78.50	87.00	59.75	39.00	44.63	43.25
25	3*	124.37	136.37	74.13	34.37	35.37	35.62
	4*	125.13	132.38	77.13	42.63	39.63	41.87
	5*	128.50	148.13	75.38	47.75	49.50	38.12
35	3*	179.75	187.75	98.50	38.25	31.25	42.00
	4*	170.00	185.63	100.13	40.38	42.38	41.25
	5*	179.75	192.25	85.25	46.88	47.00	49.35
45	3*	223.50	231.50	129.37	37.37	34.75	35.12
	4*	233.75	226.25	111.25	41.87	49.00	38.37
	5*	212.24	248.12	109.87	50.25	46.25	46.38

Note: (*) indicates critical F-value exceeded.

Table 6. Sample Means of Maximum Deviation Balance Measure for Selected Large Problems

M	N	HEURISTIC SCHEDULING PROCEDURE					
		LJN1	LJN2	LJLC	LCLR	LCGR	LCN
15	3*	.038	.033	.079	.090	.087	.085
	4	.103	.093	.112	.131	.113	.094
	5	.243	.172	.109	.171	.170	.189
25	3	.110	.143	.111	.120	.060	.109
	4*	.114	.157	.110	.096	.079	.115
	5*	.122	.190	.108	.088	.108	.099
35	3*	.112	.132	.139	.149	.035	.098
	4*	.171	.210	.195	.199	.066	.160
	5*	.167	.203	.187	.189	.078	.125
45	3*	.125	.140	.110	.157	.022	.124
	4*	.204	.247	.101	.218	.045	.154
	5*	.265	.271	.201	.239	.045	.165

Note: (*) indicates critical F-value exceeded.

Table 7. Sample Means of Average Deviation Balance Measure for Selected Large Problems

M	N	HEURISTIC RULES					
		LJN1	LJN2	LJLC	LCLR	LCGR	LCN
15	3*	.025	.023	.050	.060	.057	.057
	4	.061	.056	.069	.074	.069	.049
	5*	.124	.098	.062	.079	.100	.087
25	3*	.060	.085	.077	.081	.040	.075
	4*	.063	.083	.063	.054	.041	.052
	5*	.063	.087	.054	.048	.067	.052
35	3*	.077	.090	.060	.101	.023	.067
	4*	.089	.097	.081	.105	.035	.086
	5*	.072	.088	.075	.074	.035	.062
45	3*	.086	.095	.074	.107	.015	.085
	4*	.104	.125	.061	.111	.025	.081
	5*	.109	.112	.090	.099	.025	.073

Note: (*) indicates critical F-value exceeded.

Table 8. Sample Means of Extended-Limit-Index Balance Measure for Selected Large Problems

M	N	HEURISTIC RULE					
		LJN1	LJN2	LJLC	LCLR	LCGR	LCN
	3	.000	.000	.008	.017	.017	.013
15	4	.020	.007	.031	.046	.029	.014
	5*	.131	.081	.025	.083	.080	.098
	3*	.022	.031	.021	.047	.000	.025
25	4	.020	.062	.035	.023	.003	.020
	5*	.034	.099	.035	.012	.020	.025
	3*	.013	.036	.011	.053	.000	.020
35	4*	.075	.111	.056	.102	.000	.068
	5*	.079	.115	.072	.083	.000	.038
	3*	.030	.043	.020	.060	.000	.030
45	4*	.111	.150	.029	.121	.000	.060
	5*	.169	.175	.106	.143	.000	.072

Note: (*) indicates critical F-value exceeded.

Table 9. Average Total Change-over Costs for the Parallel Processor Problem in which M = 40, N = 4 and Variable D.

D	HEURISTIC SCHEDULING PROCEDURES					
	LJN1	LJN2	LJLC	LCLR	LCGR	LCN
4%	198.0	210.0	112.0	44.0	41.0	40.0
6%	204.0	204.0	110.0	43.0	43.0	41.0
8%	204.0	201.0	107.0	40.0	43.0	42.0
10%	203.0	214.0	102.0	40.0	43.0	41.0
12%	200.0	224.0	98.0	41.0	43.0	43.0
14%	199.0	221.0	105.0	45.0	43.0	42.0

APPENDIX B

RESULTS FOR SELECTED PROBLEMS FOR WHICH OPTIMUM SOLUTIONS WERE FOUND

Table 10. Optimal, Maximum Regret and Heuristic Total Change-over Cost Results

M	N		SCHEDULING PROCEDURES								
			OPT(UNC)	OPT(C)	MAXIMUM REGRET (1st Pass)	LJN1	LJN2	LJLC	LCLR	LCGR	LCN
			10	12	19	36	56	40	16	22	19
			8	8	11	43	54	62	30	22	22
10	2		7	9	13	43	54	41	18	17	29
			14	17	22	68	77	64	29	31	30
			11	11	20	51	65	45	27	21	26
		Avg	10.0	11.4	17.0	48.2	61.2	50.4	24.0	22.6	25.2
			10	12	27	50	91	55	31	22	17
			8	12	18	61	86	53	34	30	35
12	2		6	9	19	60	70	50	33	30	45
			9	13	32	60	56	57	33	17	17
			9	11	19	77	62	48	38	20	30
		Avg	8.4	11.4	23.0	61.6	73.0	52.6	33.8	23.8	24.8
			13	13	14	42	49	47	41	42	35
			8	23	27	30	43	34	30	30	30
6	3		10	22	27	27	41	41	27	32	32
			15	17	18	24	36	25	24	21	24
			13	21	26	46	59	39	27	27	32
		Avg	11.8	19.2	22.4	33.8	45.6	35.6	29.8	30.4	30.8

Table 11. Optimal, Maximum Regret, and Heuristic Maximum Deviation Results

M	N		SCHEDULING PROCEDURES								
			OPT(UNC)	OPT(C)	1st PASS	LJN1	LJN2	LCLR	LCGR	LJLC	LCN
10	2		.412	.058	.044	.132	.147	.000	.015	.088	.015
			.355	.118	.118	.145	.145	.132	.013	.079	.013
			.217	.043	.000	.029	.145	.058	.029	.043	.043
			.239	.119	.045	.149	.149	.090	.015	.119	.045
			.405	.000	.054	.135	.135	.027	.081	.027	.027
		Avg	.326	.068	.052	.118	.144	.061	.031	.071	.028
12	2		.179	.119	.131	.131	.131	.083	.095	.107	.024
			1.000	.062	.136	.136	.148	.123	.025	.062	.148
			.309	.107	.143	.131	.143	.048	.036	.060	.119
			1.000	.138	.064	.100	.112	.150	.050	.063	.137
			1.000	.060	.151	.145	.145	.145	.060	.096	.004
		Avg	.698	.097	.125	.129	.136	.110	.053	.078	.091
6	3		.096	.096	.096	.161	.161	.129	.161	.097	.097
			.960	.080	.040	.160	.160	.160	.160	.120	.160
			2.000	.115	.115	.154	.154	.154	.115	.115	.115
			.692	.154	.115	.154	.154	.154	.115	.115	.115
			2.000	.125	.094	.094	.125	.094	.063	.125	.031
		Avg	1.150	.114	.092	.145	.151	.138	.123	.114	.103

Table 12. Optimum, Maximum Regret, and Heuristic Average Deviation Results

M	N		SCHEDULING PROCEDURE								
			OPT(UNC)	OPT(C)	1st PASS	LJN1	LJN2	LCLR	LOGR	LJLC	LCN
			.412	.058	.044	.132	.147	.000	.015	.088	.015
			.355	.118	.118	.145	.145	.132	.013	.079	.013
10	2		.217	.043	.000	.029	.145	.058	.029	.043	.043
			.239	.119	.045	.149	.149	.090	.015	.119	.037
			.405	.000	.054	.135	.135	.027	.081	.027	.027
		Avg	.326	.068	.052	.118	.144	.061	.031	.071	.027
			.174	.086	.125	.125	.137	.077	.089	.101	.018
			.500	.062	.136	.130	.142	.117	.019	.056	.148
12	2		.309	.083	.143	.131	.143	.048	.036	.060	.119
			.500	.125	.057	.094	.106	.144	.044	.056	.131
			.500	.037	.147	.139	.139	.139	.054	.090	.018
		Avg	.397	.079	.122	.124	.133	.105	.048	.073	.087
			.054	.054	.054	.118	.118	.075	.118	.054	.075
			.500	.053	.040	.107	.107	.107	.107	.080	.107
6	3		.666	.064	.064	.115	.115	.115	.064	.064	.064
			.449	.115	.103	.115	.115	.115	.090	.090	.090
			.666	.083	.063	.063	.083	.063	.042	.083	.021
		Avg	.467	.074	.065	.104	.108	.095	.084	.074	.072

APPENDIX C

FORTRAN V CODE FOR UNCONSTRAINED OPTIMAL, BALANCED OPTIMAL, AND MAXIMUM REGRET SCHEDULING

```
      INTEGER CCOST,FINDC,CDIMEN,THETA,CHOLD,FLAG1,FLAG2
      COMMON IREED,ISEED,ICAP,JCAP,SID
      READ(5,99)M,N,NPRIME,NTYPE,LOOK,IOPT,IDUE,IBAL
   99 FORMAT( )
      DIMENSION CCOST(25,25,5),CHOLD(25,25,5),FINDC(25,25),
     1KREDCD(25,25),NODE(25,25,50),LBOUND(50),KHOLD(25,25),
     2MCHECK(20,18),MP(15,15),NTOT(5),IP(15),NIP(15),JTOT(5)
      CDIMEN=M+2*N
      INFIN=999
      MACHEN=0
      ICOUNT=0
      IPAIRS=M+N-2
      NFLAG=0
      JEND=M+3
      ITER=1
      KIP=1
      CALLLOAD(NTYPE,M,N,CCOST,KREDCD,FINDC,IP,CHOLD)
      NNODE=1
      LCOST=INFIN
  190 FORMAT(1HO,'****************************************')
      CALL REDUCE(CHOLD,CDIMEN,N,CDIMEN,CDIMEN,KREDCD,ISUM)
      LBOUND(NNODE)=ISUM
      NNODE=NNODE+1
      LBOUND(NNODE)=ISUM
  301 CALL ALTER(LOOK,NFLAG,NODE,NNODE,KREDCD,M,N,CHOLD,FINDC,THETA,
     XMROW,MCOL,ICOUNT)
      IF(THETA.EQ.(-1)) GO TO 1305
      ICOUNT=ICOUNT+1
      NNEXT=NNODE+1
      LBOUND(NNEXT)=LBOUND(NNODE)
      DO 501 I=1,CDIMEN
      DO 501 J=1,CDIMEN
      NODE(I,J,NNEXT)=NODE(I,J,NNODE)
  501 CONTINUE
      NODE(MROW,MCOL,NNODE)=-(100+FINDC(MROW,MCOL))
      LBOUND(NNODE)=LBOUND(NNODE)+THETA
      IF(LBOUND(NNODE).GT.999)LBOUND(NNODE)=999
      KPRSSR=FINDC(MROW,MCOL)
      IF(LIGHT.EQ.1) GO TO 599
      GO TO 600
  599 DO 634 I=1,KIP
      IP(I.LE.NNODE-2) GO TO 634
      DO 633 J=1,JEND
      MCHECK(I,J)=0
  633 CONTINUE
  634 CONTINUE
      KIP=NNODE-1
      LIGHT=0
      DO 625 J=1,N
      NAP=M+1
      DO 623 I=1,NAP
      MP(J,I)=0
  623 CONTINUE
      NTOT(J)=0
  625 CONTINUE
      IF(LSTASN.NE.1) GO TO 600
      DO 628 I=1,M
      NIP(I)=0
  628 CONTINUE
      LASTASN=0
  600 MCHECK(KIP,1)=MROW
      MCHECK(KIP,2)=MCOL
      MCHECK(KIP,3)=KPRSSR
      KK=MROW+3
      MCHECK(KIP,KK)=1
      LL=MCOL+3
      MCHECK(KIP,LL)=2
  602 DO 620 J5=1,N
      DO 610 I5=1,KIP
      IF(MCHECK(I5,3).NE.J5) GO TO 610
      K=MCHECK(I5,1)
      L=MCHECK(I5,2)
      IF(MP(J5,K).NE.0) GO TO 604
      NTOT(J5)=NTOT(J5)+IP(K)
      IF(I5.EQ.KIP)FLAG1=1
      MP(J5,K)=1
```

```
604 IF(MP(J5,L).NE.0) GO TO 610
    NTOT(J5)=NTOT(J5)+IP(L)
    IF(I5.EQ.KIP)FLAG2=1
    MP(J5,L)=1
610 CONTINUE
    IF(NTOT(J5).GT.JCAP) GO TO 640
620 CONTINUE
    KIP=KIP+1
    FLAG1=0
    FLAG2=0
    IF(IPAIRS.EQ.ICOUNT) GO TO 680
    GO TO 801
640 IF(FLAG1.NE.1) GO TO 650
    NTOT(J5)=NTOT(J5)-IP(K)
    MP(J5,K)=0
    FLAG1=0
650 IF(FLAG2.NE.1) GO TO 660
    NTOT(J5)=NTOT(J5)-IP(L)
    MP(J5,L)=0
    FLAG2=0
660 DO 670 J3=1,3
    MCHECK(I5,J5)=0
670 CONTINUE
    NNODE=NNODE+1
    GO TO 1305
680 LSTASN=1
    DO 684 J=4,JEND
    JJ=J-3
    DO 683 I=1,IPAIRS
    IF(MCHECK(I,J).EQ.0) GO TO 683
    NIP(JJ)=NIP(JJ)+MCHECK(I,J)
683 CONTINUE
684 CONTINUE
    DO 685 I=1,M
    IF(NIP(I).EQ.0) GO TO 690
685 CONTINUE
    GO TO 801
690 LEFT=I
    NIP(I)=3
    DO 695 I=1,M
    IF(NIP(I).EQ.3) GO TO 695
    IBEGIN=I+3
    GO TO 700
695 CONTINUE
700 DO 705 J=1,IPAIRS
    IF(MCHECK(J,IBEGIN).EQ.0) GO TO 705
    IF(MCHECK(J,IBEGIN).EQ.1) GO TO 710
    MACH=MCHECK(J,3)
    GO TO 713
705 CONTINUE
710 MACH=MCHECK(J,3)
713 NTOT(MACH)=NTOT(MACH)+IP(LEFT)
    IF(NTOT(MACH).LE.JCAP) GO TO 715
    NTOT(MACH)=NTOT(MACH)-IP(LEFT)
    NNODE=NNODE+1
    GOBTO 1305
715 DO 720 I=1,N
    IF(NTOT(I).LT.ICAP) GO TO 728
720 CONTINUE
    GO TO 801
728 NNODE=NNODE+1
    GO TO 1305
801 CALL UPDAT1(KPRSSR,CDIMEN,MROW,MCOL,N,CHOLD)
    NNODE=NNODE+1
    NODE(MROW,MCOL,NNODE)=100+FINDC(MROW,MCOL)
    CALL UPDAT2(KPRSSR,CDIMEN,MROW,MCOL,M,N,CHOLD,NFLAG,ICOUNT,
   XNODE,NNODE)
905 IF(IPAIRS-ICOUNT)1005,1005,907
907 CALL UPDATK(NPRIME,FINDC,MROW,MCOL,CHOLD,CDIMEN,M,N,
   X  KPRSSR,KREDCD,NFLAG,NODE,NNODE,ICOUNT,KHOLD)
    CALL REDUCE(CHOLD,CDIMEN,N,CDIMEN,CDIMEN,KREDCD,ISUM)
```

```
      IPASS=0
      DO 1019 I=1,CDIMEN
      DO 1019 J=1,CDIMEN
      IF(IABS(KREDCD(I,J)).EQ.999) GO TO 1019
      LROW=I
      LCOL=J
      IF(IPASS.GT.0) GO TO 1016
      ISAVE=KREDCD(LROW,LCOL)
      KREDCD(LROW,LCOL)=999
      DO 1011 K=1,CDIMEN
      IF(IABS(KREDCD(K,LCOL)).EQ.999) GO TO 1011
      KROW=K
      LSAVE=KREDCD(K,LCOL)
      KREDCD(K,LCOL)=999
 1011 CONTINUE
      NADD=0
      INUM=0
      ISTART=LROW+1
      DO 1015 IIND=ISTART,CDIMEN
      DO 1015 JIND=1,CDIMEN
      IF(IABS(KREDCD(IIND,JIND)).EQ.999) GO TO 1015
      INUM=INUM+1
 1015 CONTINUE
      IF(INUM.GT.0) GO TO 1016
      KREDCD(KROW,LCOL)=LSAVE
      IPASS=IPASS+1
      GO TO 1019
 1016 NODE(LROW,LCOL,NNODE)=100+FINDC(LROW,LCOL)
      NADD=NADD+1
      DO 1017 J1=1,CDIMEN
      KREDCD(LROW,J1)=999
 1017 CONTINUE
      DO 1018 I1=1,CDIMEN
      KREDCD(I1,LCOL)=999
 1018 CONTINUE
      IPASS=IPASS+1
 1019 CONTINUE
      IF(NADD.EQ.2) GO TO 1059
 1021 LBOUND(NNODE)=999
      GO TO 1305
 1059 LBOUND(NNODE)=LBOUND(NNODE)+ISUM
      WRITE(6,190)
      WRITE(6,1589)ITER
      IBEGIN=M+1
      IEND=M+N
      LTOTAL=0
      LSTDUN=0
      DO 1089 I=IBEGIN,IEND
      IJOB=I
      IPRSSR=IJOB-M
      JTOT(IPRSSR)=JTOT(IPRSSR)+IP(IJOB)
      WRITE(6,1080)IPRSSR,IJOB
 1080 FORMAT(1HO,13X,'SCHEDULE',I3,3X,'=',I4)
 1085 DO 1086 J=1,CDIMEN
      IF(NODE(IJOB,J,NNODE).LE.0) GO TO 1086
      JTOT(IPRSSR)=JTOT(IPRSSR)+IP(JJOB)
      JJOB=J
 1086 CONTINUE
      WRITE(6,1087)JJOB
 1087 FORMAT(30X,I3)
      LTOTAL=LTOTAL+CCOST(IJOB,JJOB,IPRSSR)
      JFINAL=M+N+IPRSSR
      IF(JJOB.EQ.JFINAL) GO TO 1088
      IJOB=JJOB
      GO TO 1085
 1088 WRITE(6,1199)JTOT(IPRSSR)
 1199 FORMAT(35X,'PROCESSING TIME =',I4)
      JTOT(IPRSSR)=0
 1089 CONTINUE
      WRITE(6,1189)LTOTAL
 1189 FORMAT(1HO,13X,'TOTAL COST',4X,'=',I4)
      IF(LBOUND(NNODE).GT.LCOST) GO TO 1305
      LCOST=LBOUND(NNODE)
      WRITE(6,1191)LCOST
 1191 FORMAT(1HO,13X,'LCOST',9X,'=',I4)
      DO 1205 I=1,CDIMEN
```

```
      DO 1205 J=1,CDIMEN
      NODE(I,J,1)=NODE(I,J,NNODE)
 1205 CONTINUE
 1305 NEXT=0
      LIGHT=1
      NINDEX=NNODE-2
      DO 1307 K=1,NINDEX
      J=NNODE-K
      IF(LBOUND(J).GE.LCOST) GO TO 1307
      IF(NEXT.GT.0) GO TO 1307
      NEXT=J
 1307 CONTINUE
      IF(NEXT.NE.0) GO TO 1505
      WRITE(6,1491)
 1491  FORMAT(14X,'CURRENT SOLUTION IS OPTIMAL')
      GO TO 9999
 1505 ISTART=NEXT+1
      DO 1507 I=ISTART,NNODE
      LBOUND(I)=0
 1507 CONTINUE
      DO 1509 K=ISTART,NNODE
      DO 1509 I=1,CDIMEN
      DO 1509 J=1,CDIMEN
      NODE(I,J,K)=0
 1509 CONTINUE
      NFLAG=0
      NNODE=NEXT
      INDEX=N-1
      DO 1515 I=1,CDIMEN
      DO 1515 J=1,CDIMEN
      KREDCD(I,J)=CCOST(I,J,1)
      FINDC(I,J)=1
      DO 1515 K=1,INDEX
      NEXT=K+1
      IF(CCOST(I,J,NEXT)-KREDCD(I,J))1514,1514,1515
 1514 KREDCD(I,J)=CCOST(I,J,NEXT)
      FINDC(I,J)=NEXT
 1515 CONTINUE
      DO 1519 K=1,N
      DO 1519 I=1,CDIMEN
      DO 1519 J=1,CDIMEN
      CHOLD(I,J,K)=CCOST(I,J,K)
 1519 CONTINUE
      LOOP=0
      DO 1521 I=1,CDIMEN
      DO 1521 J=1,CDIMEN
      IF(NODE(I,J,NNODE).EQ.0) GO TO 1521
      IF(NODE(I,J,NNODE).GT.0) NODE(I,J,NNODE)=0
      LOOP=LOOP+1
 1521 CONTINUE
      CALL REDUCE(CHOLD,CDIMEN,N,CDIMEN,CDIMEN,KREDCD,ISUM)
      ICOUNT=0
      DO 1559 L=1,LOOP
      CALL ALTER(LOOP,NFLAG,NODE,NNODE,KREDCD,M,N,CHOLD,FINDC,THETA,
     XMROW,MCOL,ICOUNT)
 1527 IF(NODE(MROW,MCOL,NNODE))1529,1555,1555
 1529 KPRSSR=FINDC(MROW,MCOL)
      CHOLD(MROW,MCOL,KPRSSR)=-999
      KREDCD(MROW,MCOL)=-999
      DO 1539 K=1,N
      IF(IABS(KREDCD(MROW,MCOL)).LE.IABS(CHOLD(MROW,MCOL,K)))
     XGO TO 1539
 1534 KREDCD(MROW,MCOL)=CHOLD(MROW,MCOL,K)
      FINDC(MROW,MCOL)=K
 1539 CONTINUE
      GO TO 1589
 1555 KPRSSR=FINDC(MROW,MCOL)
      ICOUNT=ICOUNT+1
      NODE(MROW,MCOL,NNODE)=100+KPRSSR
      CALL UPDAT1(KPRSSR,CDIMEN,MROW,MCOL,N,CHOLD)
      CALL UPDAT2(KPRSSR,CDIMEN,MROW,MCOL,M,N,CHOLD,NFLAG,
     XICOUNT,NODE,NNODE)
      CALL UPDATK(KPRIME,FINDC,MROW,MCOL,CHOLD,CDIMEN,M,N,
     XKPRSSR,KREDCD,NFLAG,NODE,NNODE,ICOUNT,KHOLD)
 1589 CALL REDUCE(CHOLD,CDIMEN,N,CDIMEN,CDIMEN,KREDCD,ISUM)
 1559 CONTINUE
      ITER=ITER+1
```

```
1589 FORMAT(1HO,'ITERATION:',I3)
     GO TO 301
9999 STOP
     END
     SUBROUTINE LOAD(NTYPE,M,N,CCOST,KCOST,FINDC,IP,CHOLD)
     INTEGER CDIMEN,CCOST,KCOST,FINDC,CHOLD,AVGCAP
     COMMON IREED,ISEED,ICAP,JCAP,SID
     CDIMEN=M+2*N
     DIMENSION CCOST(25,25,5),KCOST(25,25),CHOLD(25,25,5)
    X,IP(15),FINDC(25,25)
  53 READ(5,56)(((CCOST(I,J,K),J=1,CDIMEN),
    XI=1,CDIMEN),K=1,N)
     READ(5,56)(IP(I),I=1,M)
  56 FORMAT( )
     DO 70 I=1,M
     MTOT=MTOT+IP(I)
  70 CONTINUE
     AVGCAP=MTOT/N
     IUPPER=SID*AVGCAP
     ILOWER=IUPPER
     ICAP=AVGCAP-ILOWER
     JCAP=AVGCAP+IUPPER
     WRITE(6,89)M
  89 FORMAT(//5X,3HM =,I3)
     WRITE(6,31)N
  31 FORMAT(5X,3HN =,I3)
     WRITE(6,32)ISEED
  32 FORMAT(5X,7HSEED = ,I6)
     WRITE(6,33)SID
  33 FORMAT(5X,21HALLOWABLE DEVIATION =,F4.2)
  35 WRITE(6,34)
  34 FORMAT(/25X,11H** CCOST **)
     DO 50 K=1,N
     WRITE(6,45)K
  45 FORMAT(//1X,8HMACHINE ,I2//)
     GO TO 50
     DO 40 I=1,CDIMEN
     WRITE(6,39)(CCOST(I,J,K),J=1,CDIMEN)
  39 FORMAT(5X,25I4)
  40 CONTINUE
  50 CONTINUE
     WRITE(6,49)
  49 FORMAT(//1X,16HPROCESSING TIMES)
     WRITE(6,41)(IP(I),I=1,CDIMEN)
  41 FORMAT(//2X,20I3)
     WRITE(6,43)ICAP,AVGCAP,JCAP
  43 FORMAT(/6X,'LOWER LIMIT =',I3,2X,'AVERAGE CAPACITY =',I3,
    X2X,'UPPER LIMIT =',I3)
     DO 70 I=1,CDIMEN
     DO 70 J=1,CDIMEN
     KCOST(I,J)=CCOST(I,J,1)
     FINDC(I,J)=1
     DO 70 K=1,INDEX
     NEXT=K+1
     IF(CCOST(I,J,NEXT)-KCOST(I,J) 60,60,70
  60 KCOST(I,J)=CCOST(I,J,NEXT)
     FINDC(I,J)=NEXT
  70 CONTINUE
     DO 80 K=1,N
     DO 80 I=1,CDIMEN
     DO 80 J=1,CDIMEN
     CHOLD(I,J,K)=CCOST(I,J,K)
  80 CONTINUE
  99 RETURN
     END
     SUBROUTINE REDUCE(CHOLD,CDIMEN,N,IROWS,ICOLS,RMATRX,ISUM)
     INTEGER RMATRX,CHOLD,CDIMEN
     DIMENSION RMATRX(25,25)CHOLD(25,25,5)
     ISUM=0
     DO 50 I=1,IROWS
     MIN=999
     DO 29 J=1,ICOLS
     IF(IABS(RMATRX(I,J))-MIN) 21,29,29
  21 MIN=RMATRX(I,J)
  29 CONTINUE
     IF(MIN.EQ.0.OR.MIN.EQ.999) GO TO 50
     ISUM=ISUM+MIN
     DO 31 L=1,N
```

```
      DO 31 K=1,ICOLS
      IF(IABS(CHOLD(I,K,L))-999)30,31,31
   30 CHOLD(I,K,L)=CHOLD(I,K,L)-MIN
      CHOLD(I,K,L)=MAXO(0,CHOLD(I,K,L))
   31 CONTINUE
      DO 49 K=1,ICOLS
      IF(IABS(RMATRX(I,K))-999)37,49,49
   37 RMATRX(I,K)=RMATRX(I,K)-MIN
   49 CONTINUE
   50 CONTINUE
      DO 80 J=1,ICOLS
      MIN=999
      DO 59 I=1,IROWS
      IF(IABS(RMATRX(I,J))-MIN) 51,59,59
   51 MIN=RMATRX(I,J)
   59 CONTINUE
      IF(MIN.EQ.0.OR.MIN.EQ.999) GO TO 80
      ISUM=ISUM+MIN
      DO 61 L=1,N
      DO 61 K=1,IROWS
      IF(IABS(CHOLD(K,J,L))-999)60,61,61
   60 CHOLD(K,J,L)=CHOLD(K,J,L)-MIN
      CHOLD(K,J,L)=MAXO(0,CHOLD(K,J,L))
   61 CONTINUE
      DO 79 L=1,IROWS
      IF(IABS(RMATRX(L,J))-999)67,79,79
   67 RMATRX(L,J)=RMATRX(L,J)-MIN
   79 CONTINUE
   80 CONTINUE
      RETURN
      END
      SUBROUTINE ALTER(LOOK,NFLAG,NODE,NNODE,KREDCD,M,N,CHOLD,FINDC,
     XTHETA,MROW,MCOL,ICOUNT)
      INTEGER THETA1,THETA2,THETA,CDIMEN,FINDC,CHOLD
      CDIMEN=M+2*N
      DIMENSION CHOLD(25,25,5),
     XFINDC(25,25),KREDCD(25,25),
     XNODE(25,25,50),IMAT(25,25,5),JMAT(25,25),
     1JFIND(25,25),LOHOL(25,25)
      THETA=-1
      MROW=0
      MCOL=0
      DO 89 I=1,CDIMEN
      DO 89 J=1,CDIMEN
      IF(KREDCD(I,J).NE.0) GO TO 89
      KREDCD(I,J)=999
      MINROW=999
      DO 19 L=1,CDIMEN
      IF(IABS(KREDCD(I,L))-MINROW) 9,19,19
    9 MINROW=KREDCD(I,L)
   19 CONTINUE
      MINCOL=999
      DO 39 K=1,CDIMEN
      IF(IABS(KREDCD(K,J))-MINCOL) 29,39,39
   29 MINCOL=KREDCD(K,J)
   39 CONTINUE
      KREDCD(I,J)=0
      THETA1=MINROW+MINCOL
      IF(THETA1.GE.999) THETA1=999
      NEXT=999
      IPRSSR=FINDC(I,J)
      LSAVE=CHOLD(I,J,IPRSSR)
      CHOLD(I,J,IPRSSR)=999
      DO 59 KPRSSR=1,N
      IF(IABS(CHOLD(I,J,KPRSSR))-NEXT) 49,59,59
   49 NEXT=CHOLD(I,J,KPRSSR)
   59 CONTINUE
      CHOLD(I,J,IPRSSR)=LSAVE
      IF(999-NEXT) 61,61,69
   61 THETA2=999
      GO TO 79
   69 THETA2=NEXT-LSAVE
      IF(THETA2.LT.0)THETA2=0
   79 ITEST=MINO(THETA1,THETA2)
      IF(I.GT.M.AND.J.GT.M)ITEST=0
      IF(ITEST.LT.THETA) GO TO 89
  199 THETA=ITEST
      MROW=I
```

```
      MCOL=J
   89 CONTINUE
      RETURN
      END
      SUBROUTINE UPDAT1(KPRSSR,CDIMEN,MROW,MCOL,N,CHOLD)
      INTEGER CHOLD,CDIMEN
      DIMENSION CHOLD(25,25,5)
      KSTOP=KPRSSR-1
      KSTART=KPRSSR+1
      IF(KSTOP-1)21,7,7
    7 DO 19 K=1,KSTOP
      DO 9  J=1,CDIMEN
      CHOLD(MROW,J,K)=999
      CHOLD(MCOL,J,K)=999
    9 CONTINUE
      DO 19 I=1,CDIMEN
      CHOLD(I,MCOL,K)=999
      CHOLD(I,MROW,K)=999
   19 CONTINUE
   21 IF(N-KSTART)41,23,23
   23 DO 39 K=KSTART,N
      DO 29 J=1,CDIMEN
      CHOLD(MROW,J,K)=999
      CHOLD(MCOL,J,K)=999
   29 CONTINUE
      DO 39 I=1,CDIMEN
      CHOLD(I,MCOL,K)=999
      CHOLD(I,MROW,K)=999
   39 CONTINUE
   41 RETURN
      END
      SUBROUTINE UPDAT2(KPRSSR,CDIMEN,MROW,MCOL,M,N,CHOLD,NFLAG,
     XICOUNT,NODE,NNODE)
      INTEGER CDIMEN,CHOLD
      DIMENSION CHOLD(25,25,5),NODE(25,25,50)
      KINTL=M+KPRSSR
      KFINAL=M+N+KPRSSR
      NLEVEL=N-1
      MARK=0
      DO 3 I=1,CDIMEN
      DO 3 J=1,CDIMEN
      IP=NODE(I,J,NNODE)-100
      IF(IP.NE.KPRSSR) GO TO 3
      MARK=MARK+1
    3 CONTINUE
      IJOB=KINTL
      NPATH =1
      DO 6 KMARK=1,MARK
      IF(NPATH.EQ.0) GO TO 6
      JJOB=0
      DO 4 J=1,CDIMEN
      IF(NODE(IJOB,J,NNODE).LE.0) GO TO 4
      JJOB=J
    4 CONTINUE
      IF(JJOB.EQ.0) GO TO 5
      IJOB=JJOB
      GO TO 6
    5 NPATH=0
    6 CONTINUE
      IF(NPATH.EQ.1.AND.JJOB.EQ.KFINAL) GO TO 89
   37 JSTART=M+N+1
      JSTOP=KFINAL-1
      LSTART=KFINAL+1
      LSTOP=M+2*N
      IF(JSTOP-JSTART)43,39,39
   39 DO 41 J=JSTART,JSTOP
      CHOLD(MCOL,J,KPRSSR)=999
   41 CONTINUE
   43 IF(LSTOP-LSTART)49,45,45
   45 DO 47 J=LSTART,LSTOP
      CHOLD(MCOL,J,KPRSSR)=999
   47 CONTINUE
   49 MSTART=M+1
      MSTOP=KINTL-1
      NSTART=KINTL+1
      NSTOP=M+N
      IF(MSTOP-MSTART)63,59,59
```

```
59 DO 61 I=MSTART,MSTOP
   CHOLD(I,MROW,KPRSSR)=999
61 CONTINUE
63 IF(NSTOP-NSTART)69,65,65
65 DO 67 I=NSTART,NSTOP
   CHOLD(I,MROW,KPRSSR)=999
67 CONTINUE
69 CHOLD(KINTL,KFINAL,KPRSSR)=999
   CHOLD(MCOL,MROW,KPRSSR)=999
70 IF(NFLAG.LT.NLEVEL) GO TO 78
   DO 73 J=1,CDIMEN
   IF(CHOLD(MROW,J,KPRSSR).NE.999) GO TO 73
   CHOLD(MCOL,J,KPRSSR)=999
73 CONTINUE
   DO 77 I=1,CDIMEN
   IF(CHOLD(I,MCOL,KPRSSR).NE.999) GO TO 77
   CHOLD(I,MROW,KPRSSR)=999
77 CONTINUE
78 DO 79 J=1,CDIMEN
   CHOLD(MROW,J,KPRSSR)=999
79 CONTINUE
   DO 81 I=1,CDIMEN
   CHOLD(I,MCOL,KPRSSR)=999
81 CONTINUE
   GO TO 99
89 DO 91 I=1,CDIMEN
   DO 91 J=1,CDIMEN
   CHOLD(I,J,KPRSSR)=999
91 CONTINUE
99 RETURN
   END
   SUBROUTINE UPDATK(NPRIME,FINDC,MROW,MCOL,CHOLD,CDIMEN,M,N,
  XKPRSSR,KREDCD,NFLAG,NODE,NNODE,ICOUNT,KHOLD)
   INTEGER CHOLD,CDIMEN,FINDC
   DIMENSION CHOLD(25,25,5),FINDC(25,25),KREDCD(25,25),
  XNODE(25,25,50),KHOLD(25,25)
   KINTL=M+KPRSSR
   KFINAL=M+N+KPRSSR
   NLEVEL=N-1
   DO 2 I=1,CDIMEN
   DO 2 J=1,CDIMEN
   IF(NODE(I,J,NNODE).LE.0) GO TO 2
   K=NODE(I,J,NNODE)-100
   IF(K.EQ.KPRSSR) GO TO 2
   DO 1 LIND=1,N
   CHOLD(J,MROW,LIND)=999
 1 CONTINUE
   KREDCD(J,MROW)=999
   FINDC(J,MROW)=999
 2 CONTINUE
   MARK=0
   DO 3 I=1,CDIMEN
   DO 3 J=1,CDIMEN
   IP=NODE(I,J,NNODE)-100
   IF(IP.NE.KPRSSR) GO TO 3
   MARK=MARK+1
 3 CONTINUE
   IJOB=KINTL
   NPATH=1
   DO 6 KMARK=1,MARK
   IF(NPATH.EQ.0) GO TO 6
   JJOB=0
   DO 4 J=1,CDIMEN
   IF(NODE(IJOB,J,NNODE).LE.0) GO TO 4
   JJOB=J
 4 CONTINUE
   IF(JJOB.EQ.0) GO TO 5
   IJOB=JJOB
   GO TO 6
 5 NPATH=0
 6 CONTINUE
   IF(NPATH.EQ.1.AND.JJOB.EQ.KFINAL) GO TO 79
   DO 10 I=1,CDIMEN
   DO 10 J=1,CDIMEN
   KHOLD(I,J)=NODE(I,J,NNODE)
10 CONTINUE
```

```
      ICHAIN=KINTL
      DO 15 ILOOP=1,M
      JJOB=0
      DO 12 J=1,CDIMEN
      IF(KHOLD(ICHAIN,J).LE.0) GO TO 12
      K=KHOLD(ICHAIN,J)-100
      IF(K.NE.KPRSSR) GO TO 12
      JJOB=J
      KHOLD(ICHAIN,J)=0
   12 CONTINUE
      IF(JJOB.EQ.0) GO TO 21
      ICHAIN=JJOB
   15 CONTINUE
   21 JCHAIN=KFINAL
      DO 25 JLOOP=1,M
      IJOB=0
      DO 23 I=1,CDIMEN
      IF(KHOLD(I,JCHAIN).LE.0) GO TO 23
      K=KHOLD(I,JCHAIN)-100
      IF(K.NE.KPRSSR) GO TO 23
      IJOB=I
      KHOLD(I,JCHAIN)=0
   23 CONTINUE
      IF(IJOB.EQ.0) GO TO 27
      JCHAIN=IJOB
   25 CONTINUE
   27 KCHECK=0
      DO 33 IIND=1,CDIMEN
      DO 33 JIND=1,CDIMEN
      IF(KCHECK.GT.0) GO TO 35
      IF(KHOLD(IIND,JIND).LE.0) GO TO 33
      K=KHOLD(IIND,JIND)-100
      IF(K.NE.KPRSSR) GO TO 33
      KCHECK=1
   33 CONTINUE
      IF(KCHECK.EQ.0.AND.NFLAG.LT.NLEVEL) GO TO 8
   35 KREDCD(ICHAIN,JCHAIN)=999
      CHOLD(ICHAIN,JCHAIN,KPRSSR)=999
      FINDC(ICHAIN,JCHAIN)=999
    8 DO 9 J=1,CDIMEN
      KREDCD(MROW,J)=999
      FINDC(MROW,J)=999
    9 CONTINUE
      DO 19 I=1,CDIMEN
      KREDCD(I,MCOL)=999
      FINDC(I,MCOL)=999
   19 CONTINUE
      DO 29 J=1,CDIMENKREDCD(MCOL,J)=CHOLD(MCOL,J,KPRSSR)
      FINDC(MCOL,J)=KPRSSR
   29 CONTINUE
      DO 31 I=1,CDIMEN
      KREDCD(I,MROW)=CHOLD(I,MROW,KPRSSR)
      FINDC(I,MROW)=KPRSSR
   31 CONTINUE
      JSTART=M+N+1
      JSTOP=KFINAL-1
      LSTART=KFINAL+1
      LSTOP=M+2*N
      IF(JSTOP-JSTART)43,39,39
   39 DO 41 J=JSTART,JSTOP
      KREDCD(MCOL,J)=999
      FINDC(MCOL,J)=999
   41 CONTINUE
   43 IF(LSTOP-LSTART)49,45,45
   45 DO 47 J=LSTART,LSTOP
      KREDCD(MCOL,J)=999
      FINDC(MCOL,J)=999
   47 CONTINUE
   49 MSTART=M+1
      MSTOP=KINTL-1
      NSTART=KINTL+1
      NSTOP=M+N
      IF(MSTOP-MSTART)63,59,59
   59 DO 61 I=MSTART,MSTOP
      KREDCD(I,MROW)=999
      FINDC(I,MROW)=999
```

```
 61 CONTINUE
 63 IF(NSTOP-NSTART)69,65,65
 65 DO 67 I=NSTART,NSTOP
    KREDCD(I,MROW)=999
    FINDC(I,MROW)=999
 67 CONTINUE
 69 KREDCD(KINTL,KFINAL)=999
    GO TO 199
 79 DO 83 J=1,CDIMEN
    KREDCD(MROW,J)=999
    FINDC(MROW,J)=999
 83 CONTINUE
    DO 84 I=1,CDIMEN
    KREDCD(I,MCOL)=999
    FINDC(I,MCOL)=999
 84 CONTINUE
    DO 91 I=1,CDIMEN
    DO 91 J=1,CDIMEN
    IF(FINDC(I,J).NE.KPRSSR) GO TO 91
    KREDCD(I,J)=999
    DO 89 K=1,N
    IF(IABS(CHOLD(I,J,K))-KREDCD(I,J))81,81,89
 81 KREDCD(I,J)=CHOLD(I,J,K)
    FINDC(I,J)=K
 89 CONTINUE
 91 CONTINUE
    NFLAG=NFLAG+1
    IF(NFLAG.LT.NLEVEL) GO TO 199
    DO 97 K=1,N
    LROW=M+K
    LCOL=M+N+K
    IF(FINDC(LROW,LCOL).EQ.999) GO TO 97
    IPRO=FINDC(LROW,LCOL)
    FINDC(LROW,LCOL)=999
    KREDCD(LROW,LCOL)=999
    CHOLD(LROW,LCOL,IPRO)=999
 97 CONTINUE
139 DO 135 I=1,CDIMEN
    DO 135 J=1,CDIMEN
    KHOLD(I,J)=0
135 CONTINUE
    DO 137 K=1,M
    KHOLD(K,K)=999
137 CONTINUE
    KLEFT=0
    DO 141 I=1,CDIMEN
    DO 141 J=1,CDIMEN
    IF(IABS(KREDCD(I,J)).EQ.999) GO TO 141
    IF(KLEFT.GT.0) GO TO 141
    KLEFT=FINDC(I,J)
141 CONTINUE
    DO 149 I=1,CDIMEN
    DO 149 J=1,CDIMEN
    IF(NODE(I,J,NNODE).LE.0) GO TO 149
    K=NODE(I,J,NNODE)-100
    IF(K.NE.KLEFT) GO TO 149
    KHOLD(I,J)=999
    DO 143 J1=1,CDIMEN
    IF(KHOLD(I,J1).NE.999) GO TO 143
    KHOLD(J,J1)=999
143 CONTINUE
    DO 147 I1=1,CDIMEN
    IF(KHOLD(I1,J).NE.999) GO TO 147
    KHOLD(I1,I)=999
147 CONTINUE
149 CONTINUE
    DO 159 I=1,CDIMEN
    DO 159 J=1,CDIMEN
    IF(KHOLD(I,J).NE.999) GO TO 159
    KREDCD(I,J)=999
    CHOLD(I,J,KLEFT)=999
    FINDC(I,J)=999
159 CONTINUE
199 IF(NPRIME.EQ.0) GO TO 299
    NMACH=0
    DO 207 K=1,N
```

```
      KMACH=0
      DO 203 I=1,CDIMEN
      DO 203 J=1,CDIMEN
      LCOLV=M+N+1
      IF(I.GT.M.AND.J.GT.LCOLV) GO TO 203
      IF(NODE(I,J,NNODE)-100.NE.K) GO TO 203
      IF(KMACH.GT.0) GO TO 203
      KMACH=1
  203 CONTINUE
      NMACH=NMACH+KMACH
  207 CONTINUE
      IF(NMACH.LT.NPRIME) GO TO 299
      DO 269 K=1,N
      KMACH=0
      DO 217 I=1,CDIMEN
      DO 217 J=1,CDIMEN
      LCOLV=M+N+1
      IF(I.GT.M.AND.J.GT.LCOLV) GO TO 217
      IF(NODE(I,J,NNODE)-100.NE.K) GO TO 217
      IF(KMACH.GT.0) GO TO 217
      KMACH=1
  217 CONTINUE
      IF(KMACH.GT.0) GO TO 269
      NFLAG=NFLAG+1
      DO 219 I=1,CDIMEN
      DO 219 J=1,CDIMEN
      I1=M+K
      J1=M+N+K
      KREDCD(I1,I)=999
      KREDCD(J,J1)=999
      CHOLD(I,J,K)=999
      IF(FINDC(I,J).NE.K) GO TO 219
      KREDCD(I,J)=999
      DO 215 K1=1,N
      IF(CHOLD(I,J,K1)-KREDCD(I,J).GT.0) GO TO 215
      KREDCD(I,J)=CHOLD(I,J,K1)
      FINDC(I,J)=K1
  215 CONTINUE
  219 CONTINUE
      IF(NFLAG.LT.NLEVEL) GO TO 269
      DO 235 I=1,CDIMEN
      DO 235 J=1,CDIMEN
      KHOLD(I,J)=0
  235 CONTINUE
      DO 237 K1=1,M
      KHOLD(K1,K1)=999
  237 CONTINUE
      KLEFT=0
      DO 241 I=1,CDIMEN
      DO 241 J=1,CDIMEN
      IF(KREDCD(I,J).EQ.999) GO TO 241
      IF(KLEFT.GT.0) GO TO 241
      KLEFT=FINDC(I,J)
  241 CONTINUE
      DO 249 I=1,CDIMEN
      DO 249 J=1,CDIMEN
      IF(NODE(I,J,NNODE).LE.0) GO TO 249
      K1=NODE(I,J,NNODE)-100
      IF(K1.NE.KLEFT) GO TO 249
      KHOLD(I,J)=999
      DO 243 J1=1,CDIMEN
      IF(KHOLD(I,J1).NE.999) GO TO 243
      KHOLD(J,J1)=999
  243 CONTINUE
      DO 247 I1=1,CDIMEN
      IF(KHOLD(I1,J).NE.999) GO TO 247
      KHOLD(I1,I)=999
  247 CONTINUE
  249 CONTINUE
      DO 259 I=1,CDIMEN
      DO 259 J=1,CDIMEN
      IF(KHOLD(I,J).NE.999) GO TO 259
      KREDCD(I,J)=999
      CHOLD(I,J,KLEFT)=999
  259 CONTINUE
  269 CONTINUE
  299 RETURN
      END
```

APPENDIX D

FORTRAN V CODE FOR HEURISTIC SCHEDULING

```
      DIMENSION CCOST(60,60,5),P(50),IP(50),LCOST(50),NCOST(5,50),
     1ISTART(5),WORK(5),IWORK(5),GO(50),TOTAL(5),NIP(5),LIGHT(5),
     2JOB(5,50),IJOBTO(5),LJOB(5,50),KSTART(5),LZ(5),JOBTOT(5),
     3JWORK(5),TOT(5),LSTDUN(5),BCOST(50),ACOST(5,50),FIRST(5)
     4COST(50,5),AVCOST(5),AVCST(5),AVBAL1(5),AVBAL2(5),AVBAL3(5),SUM(5)
      EQUIVALENCE (JOBTOT,WORK),(IJOBTO,IWORK)
      COMMON AVGCAP,IN,INRATE,KEEP,NUMBER,MTOT,IMP,N,JCAP,MOST
      COMMON BALMAT(50,5,4),BALNCE(5)
      COMMON ISEED,STD(30,5)
      INTEGER WORK,FLAG,GO,TOTAL,ASSN,COST,CDIMEN,BALNCE
      INTEGER CCOST,P,SHAPE,SHIP,AVGCAP,TOT,BCOST,ACOST,FIRST,AVCOST
      READ(5,10)M,N,ISEED,IRUNS,SID
   10 FORMAT( )
      SN=N
      NOBS=IRUNS*5
      IN=N-1
      IM=M-1
      CDIMEN=M+(2*N)
      IMP=IRUNS
      PIN=IMP
      KEEP=1
      READ(5,10)(((CCOST(I,J,K),J=1,CDIMEN),I=1,CDIMEN),
     XK=1,N)
      READ(5,10)(P(I),I=1,M)
      DO 25 I=1,M
      IP(I)=P(I)
      NIP(I)=IP(I)
      MTOT=MTOT+P(I)
   25 CONTINUE
      ITER=999
      DO 32 I=1,M
      IF(IP(I)-ITER)31,32,32
   31 MIN=I
      ITER=IP(I)
   32 CONTINUE
      LITTLE=IP(MIN)
      AVGCAP=MTOT/N
      IUPPER=STD*AVGCAP
      ILOWER=IUPPER
      ICAP=AVGCAP-ILOWER
      JCAP=AVGCAP+IUPPER
      IF(M.GT.30) GO TO 47
      DO 50 K=1,N
      WRITE(6,34)
   34 FORMAT(//5X,11H** CCOST **)
      WRITE(6,45)K
   45 FORMAT(//1X,8HMACHINE ,I2//)
      DO 40 I=1,CDIMEN
      WRITE(6,39)(CCOST(I,J,K),J=1,CDIMEN)
   39 FORMAT(5X,25I4)
   40 CONTINUE
   50 CONTINUE
   47 WRITE(6,49)
   49 FORMAT(//1X,16HPROCESSING TIMES)
      WRITE(6,41)(P(I),I=1,CDIMEN)
   41 FORMAT(//2X,20I3)
      WRITE(6,43)ICAP,AVGCAP,JCAP
   43 FORMAT(/9X,6HICAP =,I3,2X,18HAVERAGE CAPACITY =,I3,2X,6HJCAP =,I3)
      NUMBER=1
      GO TO 250
  250 DO 270 I=1,M
      IP(I)=NIP(I)
      P(I)=IP(I)
      BCOST(I)=0
      LCOST(I)=0
      GO(I)=0
  270 CONTINUE
      DO 275 I=1,N
      LIGHT(I)=0
      TOT(I)=0
      LSTDUN(I)=0
      JWORK(I)=0
      IWORK(I)=0
      JOBTOT(I)=0
      FIRST(I)=0
  275 CONTINUE
      DO 280 I=1,N
      DO 278 J=1,M
```

```
      LJOB(I,J)=0
      ACOST(I,J)=0
      JOB(I,J)=0
      KASN(I,J)=0
  278 CONTINUE
  280 CONTINUE
      IF(NUMBER.EQ.7) GO TO 999
      IF(NUMBER.EQ.6) GO TO 1200
      IF(NUMBER.EQ.5) GO TO 900
      IF(NUMBER.EQ.4) GO TO 600
      IF(NUMBER.EQ.3) GO TO 600
      IF(NUMBER.EQ.2) GO TO 440
      NRATE=0
      MCNT=0
      DO 435 I=1,N
      JN=M+I
      DO 410 K=1,M
      NCOST(I,K)=CCOST(JN,K,I)
      ACOST(I,K)=NCOST(I,K)
  410 CONTINUE
  415 DO 420 J=1,IM
      IF(NCOST(I,J).GT.NCOST(I,J+1)) GO TO 420
      IDUP=NCOST(I,J)
      NCOST(I,J)=NCOST(I,J+1)
      NCOST(I,J+1)=IDUP
  420 CONTINUE
      KSTART(I)=NCOST(I,M)
      DO 430 J=1,M
      IF(KSTART(I).NE.ACOST(I,J)) GO TO 430
      IF(GO(J).EQ.1) GO TO 425
      JOB(I,1)=J
      GO(J)=1
      JOBTOT(I)=P(J)
      IP(J)=0
      P(J)=0
      GO TO 435
  425 NCOST(I,M)=999
      ACOST(I,J)=999
      GO TO 415
  430 CONTINUE
  435 CONTINUE
      IT=1
      LEFT=M-N
      GO TO 445
  440 NRATE=0
      LEFT=M
      MCNT=0
      IT=0
  445 IT=IT+1
  447 IF(NUMBER.EQ.2) GO TO 480
      DO 450 I=1,N
      IJOBTO(I)=JOBTOT(I)
  450 CONTINUE
      IF(IT.EQ.1) GO TO 480
      DO 460 I=1,IN
      IF(IJOBTO(I).LE.IJOBTO(I+1)) GO TO 460
      MDUD=IJOBTO(I)
      IJOBTO(I)=IJOBTO(I+1)
      IJOBTO(I+1)=MDUD
  460 CONTINUE
      IDUN=IJOBTO(N)
  470 DO 475 J=1,N
      IF(IDUN.NE.JOBTOT(J)) GO TO 475
      SHIP=J
      GO TO 480
  475 CONTINUE
  480 DO 490 I=1,IM
      IF(IP(I).LE.IP(I+1)) GO TO 490
      IDUT=IP(I)
      IP(I)=IP(I+1)
      IP(I+1)=IDUT
  490 CONTINUE
      IDUT=IP(M)
  510 DO 520 J=1,M
      IF(IDUT.NE.P(J)) GO TO 520
      SHARE=J
      IF(NUMBER.EQ.2.AND.IGNITE.EQ.1) GO TO 530
      IF(NUMBER.EQ.2) GO TO 511
```

```
      IF(IT.NE.1) GO TO 530
  511 DO 512 I=1,N
      LI=N+I
      LCOST(I)=CCOST(LT,SHAPE,I)
      BCOST(I)=LCOST(I)
  512 CONTINUE
      DO 515 I=1,IN
      IF(LCOST(I).GE.LCOST(I+1)) GO TO 515
      IJIP=LCOST(I)
      LCOST(I)=LCOST(I+1)
      LCOST(I+1)=IJIP
  515 CONTINUE
      IMIP=LCOST(N)
      DO 518 I=1,N
      IF(IMIP.NE.BCOST(I)) GO TO 518
      IF(NUMBER.EQ.2) GO TO 516
      MCNT=1
      SHIP=I
      GO TO 530
  516 MCNT=MCNT+1
      SHIP=I
      GO TO 530
  518 CONTINUE
  520 CONTINUE
  530 JOB(SHIP,IT)=SHAPE
      GO(SHAPE)=1
      JOBTOT(SHIP)=JOBTOT(SHIP)+P(SHAPE)
      TOTAL(SHIP)=JOBTOT(SHIP)
      IF(JOBTOT(SHIP).LE.JCAP) GO TO 540
      IF(MCNT.EQ.N) GO TO 540
      JOB(SHIP,IT)=0
      JOBTOT(SHIP)=JOBTOT(SHIP)-P(SHAPE)
      TOTAL(SHIP)=JOBTOT(SHIP)
  535 IP(M)=0
      LEFT=LEFT-1
      IF(LEFT.EQ.0) GO TO 538
      LIGHT(SHIP)=1
      GO TO 480
  538 TOTAL(SHIP)=JOBTOT(SHIP)
      IF(NUMBER.EQ.2) GO TO 536
      JOBTOT(SHIP)=0
      MCNT=MCNT+1
      LEFT=MAT
  536 DO 485 I=1,M
      IP(I)=P(I)
  485 CONTINUE
      IGNITE=0
      LIGHT(SHIP)=1
      GO TO 447
  540 P(SHAPE)=1
      DO 545 I=1,M
      IP(I)=P(I)
  545 CONTINUE
      IF(NUMBER.EQ.2) GO TO 547
      IREMAN=JCAP-JOBTOT(SHIP)
      IF(IREMAN.LT.LITTLE)JOBTOT(SHIP)=0
      GO TO 546
  547 LEFT=M-IT
      MAT=LEFT
      JX=M
      GO TO 548
  546 JX=M-N+1
      LEFT=M-N-IT+1
      MAT=LEFT
  548 IF(IT.EQ.JX) GO TO 550
      IGNITE=1
      GO TO 445
  550 WRITE(6,53)NUMBER
   53 FORMAT(//5X,24H********** HEURISTIC ,I3,13H **********//)
      DO 580 I=1,N
      K=1
      WRITE(6,59)I
   59 FORMAT(/6X,9HSCHEDULE I,3H = )
      DO 565 J=1,M
      IF(JOB(I,J).EQ.0) GO TO 565
      WRITE(6,61)JOB(I,J)
   61 FORMAT(10X,24X,I3)
      LOOB(I,K)=JOB(I,J)
```

```
      KAN=LJOB(I,1)
      TOT(I)=TOT(I)+NIP(KAN)
      JOB(I,J)=0
      GO TO 570
  565 CONTINUE
  570 DO 575 J=1,M
      IF(JOB(I,J).EQ.0) GO TO 575
      WRITE(6,63)JOB(I,J)
   63 FORMAT(25X,I3)
      K=K+1
      LJOB(I,K)=JOB(I,J)
      KAN=LJOB(I,K)
      TOT(I)=TOT(I)+NIP(KAN)
  575 CONTINUE
  580 CONTINUE
      DO 585 I=1,N
      WRITE(6,65)I,TOT(I)
   65 FORMAT(/5X,23HTOTAL WORK ON PROCESSOR,I4,2X,3H = ,I5)
  585 CONTINUE
      CALL DEVATE(TOT)
      DO 589 I=1,N
      ITHREE=M+N+I
      IONE=M+I
      ITWO=LJOB(I,1)
      COST(KEEP,NUMBER)=COST(KEEP,NUMBER)+CCOST(IONE,ITWO,I)
      DO 587 J=1,M
      IONE=LJOB(I,J)
      ITWO=LJOB(I,J+1)
      IF(ITWO.EQ.0) GO TO 588
      COST(KEEP,NUMBER)=COST(KEEP,NUMBER)+CCOST(IONE,ITWO,I)
  587 CONTINUE
  588 COST(KEEP,NUMBER)=COST(KEEP,NUMBER)+CCOST(IONE,ITHREE,I)
  589 CONTINUE
      NUMBER=NUMBER+1
      GO TO 250
  600 ASSN=1
      LAST=0
  605 DO 630 I=1,N
      JN=M+I
      DO 610 K=1,M
      NCOST(I,K)=CCOST(JN,K,I)
      ACOST(I,K)=NCOST(I,K)
  610 CONTINUE
  612 DO 615 J=1,IM
      IF(NCOST(I,J).GT.NCOST(I,J+1)) GO TO 615
      IDUP=NCOST(I,J)
      NCOST(I,J)=NCOST(I,J+1)
      NCOST(I,J+1)=IDUP
  615 CONTINUE
      ISTART(I)=NCOST(I,M)
      DO 625 J=1,M
      IF(ISTART(I).NE.ACOST(I,J)) GO TO 625
      IF(GO(J).EQ.1) GO TO 622
      JOB(I,ASSN)=J
      GO(J)=1
      LSTJOB(I)=J
      WORK(I)=P(J)
      IWORK(I)=WORK(I)
      GO TO 630
  622 NCOST(I,M)=999
      ACOST(I,J)=999
      GO TO 612
  625 CONTINUE
  630 CONTINUE
      LEFT=N-N
      MAT=LEFT
  645 DO 650 I=1,IN
      IF(NUMBER.EQ.4) GO TO 647
      IF(IWORK(I).LE.IWORK(I+1)) GO TO 650
      GO TO 649
  647 IF(IWORK(I).GE.IWORK(I+1)) GO TO 650
  649 IDUP=IWORK(I)
      IWORK(I)=IWORK(I+1)
      IWORK(I+1)=IDUP
  650 CONTINUE
      NDUM=IWORK(M)
  670 DO 675 I=1,N
      IF(NDUM.NE.WORK(I)) GO TO 675
```

```
      MACH=I
      GO TO 679
  675 CONTINUE
  679 IND=LSTDUN(MACH)
  687 DO 688 I=1,M
      LCOST(I)=CCOST(IND,I,MACH)
      BCOST(I)=LCOST(I)
  688 CONTINUE
  690 DO 691 I=1,M
      IF(GO(I).EQ.0) GO TO 691
      LCOST(I)=999
      BCOST(I)=999
  691 CONTINUE
  692 DO 700 J=1,IM
      IF(LCOST(J).GE.LCOST(J+1)) GO TO 700
      MDUP=LCOST(J)
      LCOST(J)=LCOST(J+1)
      LCOST(J+1)=MDUP
  700 CONTINUE
      JDUM=LCOST(M)
      FLAG=M
  720 DO 730 J=1,M
      IF(JDUM.NE.BCOST(J)) GO TO 730
      IF(GO(J).EQ.1) GO TO 728
  725 JOB(MACH,ASSN+1)=J
      IS=J
      GO TO 735
  728 LCOST(M)=999
      BCOST(J)=999
      LEFT=LEFT-1
      IF(LEFT.EQ.0) GO TO 736
      GO TO 692
  730 CONTINUE
  735 WORK(MACH)=WORK(MACH)+P(IS)
      JWORK(MACH)=WORK(MACH)
      IF(WORK(MACH).LE.JCAP) GO TO 745
      IF(LAST.EQ.N) GO TO 745
      LCOST(FLAG)=999
      BCOST(IS)=999
      JOB(MACH,ASSN+1)=0
      WORK(MACH)=WORK(MACH)-P(IS)
      LEFT=LEFT-1
      IF(LEFT.NE.0) GO TO 692
  736 JWORK(MACH)=WORK(MACH)
      WORK(MACH)=0
      LEFT=MAT
      LAST=LAST+1
      DO 738 I=1,N
      IWORK(I)=WORK(I)
  738 CONTINUE
      GO TO 645
  745 IF(ASSN.EQ.M-N) GO TO 560
      LSTDUN(MACH)=IS
      GO(IS)=1
      DO 750 I=1,N
      IWORK(I)=WORK(I)
      JWORK(I)=IWORK(I)
  750 CONTINUE
      LEFT=M-N-ASSN
      MAT=LEFT
      ASSN=ASSN+1
      GO TO 645
  900 NMATE=0
      LAST=1
      ASSN=1
      LEFT=M
  902 DO 905 J=1,IM
      IF(IP(J).LE.IP(J+1)) GO TO 905
      ITIP=IP(J)
      IP(J)=IP(J+1)
      IP(J+1)=ITIP
  905 CONTINUE
      ISIP=IP(1)
      DO 910 J=1,M
      IF(ISIP.NE.P(J)) GO TO 910
      SMAPE=J
      GO TO 920
  910 CONTINUE
```

```
  920 DO 930 J=1,N
      JM=M+J
      IF(FIRST(J).EQ.1) GO TO 925
      LCOST(J)=CCOST(JM,SHAPE,J)
      GO TO 930
  925 JOAT=LSTDUN(J)
      LCOST(J)=CCOST(JOAT,SHAPE,J)
  930 CONTINUE
      IMIT=999
  938 DO 940 J=1,N
      IF(LCOST(J)-IMIT)935,940,940
  935 SHIP=J
      IMIT=LCOST(J)
  940 CONTINUE
      IF(IMIT.EQ.999) GO TO 976
      IMIT=999
      JOB(SHIP,ASSN)=SHAPE
      JWORK(SHIP)=JWORK(SHIP)+P(SHAPE)
      IF(JWORK(SHIP).GT.JCAP) GO TO 960
      LEFT=LEFT-1
      FIRST(SHIP)=I
      LSTDUN(SHIP)=SHAPE
      IF(LEFT.EQ.0) GO TO 550
      ASSN=ASSN+1
      P(SHAPE)=0
      IP(M)=0
      GO TO 902
  976 IMIT=999
      DO 981 J=1,N
      IF(JWORK(J)-IMIT)979,981,981
  979 SHIP=J
      IMIT=JWORK(J)
  981 CONTINUE
      JOB(SHIP,ASSN)=SHAPE
      JWORK(SHIP)=JWORK(SHIP)+P(SHAPE)
      LSTDUN(SHIP)=SHAPE
  992 LEFT=LEFT-1
      IF(LEFT.EQ.0) GO TO 550
      ASSN=ASSN+1
      P(SHAPE)=0
      IP(M)=0
      GO TO 902
  960 JWORK(SHIP)=JWORK(SHIP)-P(SHAPE)
      JOB(SHIP,ASSN)=0
      LCOST(SHIP)=999
      GO TO 938
 1200 LEFT=M
 1270 MAT=LEFT
 1275 DO 1310 K=1,N
      MR=M+K
      LOOK=LSTDUN(K)
      IF(LOOK.EQ.0) GO TO 1285
      DO 1280 J=1,M
      BCOST(J)=CCOST(LOOK,J,K)
      IF(KASN(K,J).EQ.1)BCOST(J)=999
 1280 CONTINUE
      GO TO 1292
 1285 DO 1290 J=1,M
      BCOST(J)=CCOST(MR,J,K)
      IF(KASN(K,J).EQ.1)BCOST(J)=999
 1290 CONTINUE
 1292 ITER=999
      DO 1300 JJ=1,M
      IF(BCOST(JJ)-ITER)1295,1300,1300
 1295 LIGHT(K)=BCOST(JJ)
      FIRST(K)=JJ
      JM=JJ+1
      ITER=BCOST(JJ)
 1300 CONTINUE
 1310 CONTINUE
 1315 ITER=999
      DO 1330 I=1,N
      IF(LIGHT(I)-ITER)1320,1330,1330
 1320 MINE=I
      ITER=LIGHT(I)
 1330 CONTINUE
      IF(LAST.EQ.1)SAVE=FIRST(MINE)
      LAST=0
```

```
      SHAPE=FIRST(MIN)
      JOBTOT(MIN)=JOBTOT(MIN)+NIP(SHAPE)
      IF(JOBTOT(MIN).GT.JCAP) GO TO 1350
      IP(SHAPE)=0
      JOB(MIN,ASSN)=SHAPE
 1333 DO 1345 K=1,M
      IF(IP(K).NE.0) GO TO 1338
      DO 1335 I=1,N
      KASN(I,K)=1
 1335 CONTINUE
      GO TO 1345
 1338 DO 1340 I=1,N
      KASN(I,K)=0
 1340 CONTINUE
 1345 CONTINUE
      LEFT=LEFT-1
      IF(LEFT.EQ.0) GO TO 550
      ASSN=ASSN+1
      LSTJOB(MIN)=SHAPE
      MACH=N
      LAST=1
      GO TO 1270
 1350 JOBTOT(MIN)=JOBTOT(MIN)-NIP(SHAPE)
      LIGHT(MIN)=999
      LOT=FIRST(MIN)
      KASN(MIN,LOT)=1
      MACH=MACH-1
      IF(MACH.EQ.0) GO TO 1360
      GO TO 1315
 1360 MAT=MAT-1
      IF(MAT.EQ.0) GO TO 1380
      GO TO 1275
 1380 ITER=999
      DO 1400 I=1,N
      IF(JOBTOT(I)-ITER)1390,1400,1400
 1390 MIN=I
      ITER=JOBTOT(I)
 1400 CONTINUE
      JOB(MIN,ASSN)=SAVE
      IP(SAVE)=0
      GO TO 1333
  999 NUMBER=0
      KEEP=KEEP+1
      MTOT=0
      IGNITE=0
      IRUNS=IRUNS-1
      IF(IPUNS.NE.0) GO TO 100
 1000 IDUP=0
      WRITE(6,1070)
      WRITE(6,29)M
   29 FORMAT(//5X,3HM =,I3)
      WRITE(6,31)N
   31 FORMAT(5X,3HN =,I3)
      WRITE(6,32)ISEED
   32 FORMAT(5X,7HSEED = ,I6)
      WRITE(6,33)SID
   33 FORMAT(5X,21HALLOWABLE DEVIATION =1F4.2)
      DO 1020 I=1,5
      DO 1010 J=1,IMP
      IDUP=IDUP+COST(J,I)
      IDUM=IDUM+BALMAT(J,I,1)
      JDUM=JDUM+BALMAT(J,I,2)
      KDUM=KDUM+BALMAT(J,I,3)
 1010 CONTINUE
      AVCOST(I)=IDUP/IMP
      AVBAL1(I)=IDUM/IMP
      AVBAL2(I)=JDUM/IMP
      AVBAL3(I)=KDUM/IMP
      IDUP=0
      IDUM=0
      JDUM=0
      KDUM=0
 1020 CONTINUE
      WRITE(6,1030)
 1030 FORMAT(30X,9HCOST(I,J))
      DO 1050 I=1,IMP
      WRITE(6,1040)(COST(I,J),J=1,6)
 1040 FORMAT(20X,6I3)
```

```
1050 CONTINUE
     WRITE(6,1060)(AVCOST(I),I=1,6)
1060 FORMAT(6X,15HAVERAGE COST = ,6I5)
     WRITE(6,1070)
1070 FORMAT(10X,'**********************************************'/)
     WRITE(6,1080)
1080 FORMAT(13X,17HMAXIMUM DEVIATION,10X,17HAVERAGE DEVIATION,
    X9X,19HMAX EXTENDED LIMIT,9X,18HAVG EXTENDED LIMIT/)
     DO 1100 I=1,IMP
     WRITE(6,1090)(BALMAT(I,J,1),J=1,6),(BALMAT(I,J,2),J=1,6),
    X(BALMAT(I,J,3),J=1,6),(BALMAT(I,J,4),J=1,6)
1090 FORMAT(12X,6F4.2,7X,6F4.2,7X,6F4.2,7X,6F5.4)
1100 CONTINUE
9999 END
     SUBROUTINE DEVATE(BALMAT,TOT)
     COMMON AVGCAP,IN,NRATE,KEEP,NUMBER,MTOT,IMP,N,JCAP
     DIMENSION BALMAT(50,5,3),BALNCE(10),TOT(10)
     INTEGER BALNCE,BALMAT,TOT
     IF(NRATE.EQ.0) GO TO 30
     DO 10 I=1,N
     BALNCE(I)=SQRT((TOT(I)-AVGCAP)**2)
     MOST=MOST+BALNCE(I)
  10 CONTINUE
     DO 20 I=1,IN
     IF(BALNCE(I).LE.BALNCE(I+1)) GO TO 20
     IDUP=BALNCE(I)
     BALNCE(I)=BALNCE(I+1)
     BALNCE(I+1)=IDUP
  20 CONTINUE
     BALMAT(KEEP,NUMBER,1)=(BALNCE(N)/AVGCAP)
     BALMAT(KEEP,NUMBER,2)=((MOST/N)/AVGCAP)
     GO TO 40
  30 BALMAT(KEEP,NUMBER,1)=0
     BALMAT(KEEP,NUMBER,2)=0
     BALMAT(KEEP,NUMBER,3)=0
     GO TO 60
  40 IDUM=JCAP-AVGCAP
     IDIP=BALNCE(N)-IDUM
     BALMAT(KEEP,NUMBER,3)=IDIP/AVGCAP
  60 RETURN
     END
```

BIBLIOGRAPHY

1. Ashour, S., Sequencing Theory, Springer-Verlag, New York, 1972.

2. Ashour, S., Vega, J. F., and Parker, R. G., "A Heuristic Algorithm for Traveling Salesman Problems," Transportation Research, 6 (1972), 187-195.

3. Conway, R. W., Maxwell, W. L., and Miller, L. W., Theory of Scheduling, Addison-Wesley, Reading, Massachusetts (1967).

4. Elmaghraby, S. E., "The Sequencing of N Jobs of M Parallel Processors," Research Memorandum, North Carolina State University, January, 1968.

5. Eilon, S., and Christofides, N., "The Loading Problem," Management Science, 17 (1971), 5 (January), 259-273.

6. Gavett, J. W., "Three Heuristic Rules for Sequencing Jobs to a Single Production Facility," Management Science, 11 (1965), 8 (June), 166-176.

7. Mize, J. H., White, C. R., and Brooks, G. H., Operations Planning and Control, Prentice-Hall, Englewood Cliffs, New Jersey, 1971.

8. Gere, W. S., "Heuristics in Job Shop Scheduling," Management Science, 13 (1966), 3 (November), 167-190.

9. Sisson, R. L., Sequencing Theory, "Progress in Operations Research," Vol. 1, Ackoff, R. L., ed., Ch. 7, John Wiley, 1961.

10. Deane, R. D. and Moodie, C. L., "A Dispatching Methodology for Balancing Workload Assignments in a Job Shop Production Facility," American Institute of Industrial Engineers, 4(1972), 4(December), 277-282.

11. Greenberg, I., "Application of the Loading Algorithm to Balance Workloads," American Institute of Industrial Engineers, 4 (1972), 4 (December), 337-339.

12. Marsh, J. M., "Scheduling on Parallel Processors," Doctoral Dissertation, Georgia Institute of Technology (1973).

13. Pierce, J. F. and Hatfield, D. J., "Production Sequencing by Combinatorial Programming," Chapter 17 of J. F. Pierce, Operations Research and the Design of Management Information Systems, Technical Association of the Pulp and Paper Industry, New York (1967).

14. Little, J. D. C., Murty, K. G., Sweeny, D. W., and Karel, C., An Algorithm for the Traveling Salesman Problem," Operations Research, 9 (1961), 6 (November), 841-848.

15. Lockett, A. G. and Muhlemann, A. P., "A Scheduling Problem Involving Sequence-Dependent Changeover Times," Operations Research, 20 (1972), 4 (July-August), 895-902.

16. Irasterza, J. C., "A Control and Balancing Methodology for the Parallel Processing Shop," Doctoral Dissertation, Georgia Institute of Technology (1973).